UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

ARQUITECTURA DE SOFTWARE APLICADA A UN SISTEMA EXPERTO DE EVALUACIÓN DE RIESGOS NATURALES

TESIS

PARA OBTENER EL TÍTULO DE

INGENIERO CIVIL

PRESENTA:

CÉSAR MENDOZA CANALES

DIRECTOR DE TESIS:
DR. MARIO GUSTAVO ORDAZ SCHROEDER
INSTITUTO DE INGENIERÍA, UNAM

MÉXICO, D.F. OCTUBRE, 2014.

VNIVERSIDAD NACIONAL
AVPNºMA DE
MEXICO

DIVISIÓN DE INGENIERÍAS CIVIL Y GEOMÁTICA
COMITÉ DE TITULACIÓN
FING/DICyG/SEAC/UTIT/95/2014

Señor
CESAR MENDOZA CANALES
Presente

En atención a su solicitud me es grato hacèr de su conocimiento el tema que propuso el profesor DR. MARIO GUSTAVO ORDAZ SCHROEDER que aprobó este Comité, para que lo desarrolle usted conforme a la opción I. "Titulación mediante tesis o tesina y examen profesional", para obtener su título en INGENIERIA CIVIL

"ARQUITECTURA DE SOFTWARE APLICADA A UN SISTEMA EXPERTO DE EVALUACIÓN DE RIESGOS NATURALES"

INTRODUCCIÓN
I. EVALUACIÓN DE RIESGOS NATURALES
II. SISTEMAS EXPERTOS
III. ARQUITECTURA DE SOFTWARE
IV. DISEÑO DE LA ARQUITECTURA
V. IMPLEMENTACIÓN DE LA ARQUITECTURA
VI. CONCLUSIONES
BIBLIOGRAFÍA

Ruego a usted cumplir con la disposición de la Dirección General de la Administración Escolar en el sentido de que se imprima en lugar visible de cada ejemplar de la tesis el Título de ésta.

Asimismo le recuerdo que la Ley de Profesiones estipula que deberá prestar servicio social durante un tiempo mínimo de seis meses como requisito para sustentar Examen Profesional

Atentamente
"POR MI RAZA HABLARA EL ESPÍRITU"
Cd. Universitaria a 3 de Septiembre de 2014
EL PRESIDENTE DEL COMITÉ

M. EN I. JOSÉ LUIS TRIGOS SUÁREZ
JLTS/MTH

Agradecimientos

Con este trabajo culmino una etapa de mi vida académica, a la cual he llegado gracias al apoyo de tantas personas a quienes aprecio y amo; por ende les agradezco con humildad y júbilo:

A Dios y a la virgen de Guadalupe que me regalaron la vida.

Al Espíritu Santo por iluminarme, darme motivación y fortaleza en cada día dedicado a este trabajo.

A mis padres Emma y José por su amor infinito. A mi mamá por llenarme de bendiciones con sus oraciones. A mi papá por hacer de mí un niño feliz y un hombre de bien.

A mi esposa Mery y mi hija Zómer quienes me aman incondicionalmente. Son mi fuente de Fe. A mi suegra Zonia por cuidar con tanto amor a mi hija.

A mi tía Bertha y tío Alfredo quienes me dieron un hogar y el trato de un hijo durante mi vida universitaria.

A mi hermano Alejandro por ser mi ejemplo de tenacidad y fe.

A mis tías: Chanita, Dalida, Lelos, Tella y Nely quienes me alentaban a estudiar.

A mis tíos: Alberto, Angel, Apolinar, Lorenzo, Martín y Servando quienes me inspiraban para llevar una vida recta.

A todos mis profesores que sembraron en mí la pasión por aprender, en especial al profesor Armando y al maestro Santiago que me llevaron por el sendero correcto.

A mis hermanos de comunidad por sus oraciones y brindarme la confianza para perseverar.

Al Dr. Daniel Rojas, M.I. Benjamín Huerta y al Ing. Javier Aguilar por su valiosa amistad y su apoyo constante durante mis redacciones.

A mis cómplices de estudio Apolinar Sánchez Juárez, Israel A. Sánchez Ortiz y Guillermo Monter Vergara, con quien compartí miles de horas de aprendizaje en las aulas la UNAM, nuestra alma mater.

A mis sinodales por regalarme el honor de ser mi jurado: Dra. Amparo López Gaona, M.I. José Luis Trigos Suárez, Dr. Eduardo Reinoso Angulo y Dr. Nicolás Kemper Valverde.

Y mi profundo agradecimiento al orquestador de este trabajo, al Dr. Mario Ordaz Schroeder director de mi tesis, por llevarme hasta buen puerto y brindarme los elementos necesarios para lograrlo, pero sobre todo, por confiar en mí a pesar del largo caminar que me llevó concluir esta meta.

INDICE

ÍNDICE

RESUMEN

CAPRA (Comprehensive Approach to Probabilistic Risk Assessment) es un sistema experto de evaluación de riesgos naturales impulsado por el Banco Mundial para fortalecer la capacidad institucional en evaluación del riesgo de desastre.

El ciclo de vida de su software está limitado debido a sus actuales características, mismas que impiden la migración hacia nuevas tecnologías de cómputo móvil y en la nube. La arquitectura y estructura de código requieren ser actualizadas para evitar el problema inminente de quedar obsoletas.

Este trabajo presenta el diseño e implementación de una arquitectura de software propuesta para el sistema CAPRA; también se ejecuta un proceso para la reestructuración de su código hacia la nueva arquitectura. Con esto, se pretende establecer un marco de desarrollo que permita su evolución y nuevos desarrollos.

INTRODUCCIÓN

ANTECEDENTES

Los desastres naturales son eventos que involucran a sismos, tsunamis, erupciones volcánicas, inundaciones y huracanes, que ocasionan pérdidas irreparables de vidas humanas. Adicionalmente, conllevan pérdidas económicas y materiales cuantificadas hasta en decenas de miles de millones de dólares por evento.

La ocurrencia de eventos naturales es inevitable; sin embargo, sí es posible mitigar y prevenir que un evento de este tipo se convierta en un desastre natural. Como primer paso, se requiere cuantificar de manera razonable los riesgos a los que se expone la estructura, ciudad o región y a partir de ello tomar decisiones.

La cuantificación del riesgo se realiza con el apoyo de piezas de software consideradas sistemas expertos. Este tipo de sistemas expertos existen en el ámbito de la industria privada, aunque por su parte, el Banco Mundial impulsa una iniciativa que busca fortalecer la capacidad institucional en evaluación del riesgo de desastre ofreciendo un sistema experto de libre uso que tiene por nombre CAPRA.

Con esa visión, el uso de CAPRA debe ser fomentado como parte de los procesos de diseño en la ingeniería civil y en la toma de decisiones de gobiernos de todos los niveles durante la planeación y construcción de zonas urbanas, industriales e infraestructura de comunicaciones.

Sin embargo, CAPRA es una implementación de software que está próxima a ser obsoleta, debido a que en las nuevas tecnologías de cómputo están obligando a dejar atrás aplicaciones de escritorio y migrar hacia aplicaciones para dispositivos móviles y cómputo en la nube.

Para evitar la obsolescencia y desaparición de este sistema experto, es necesaria la evolución de CAPRA como elemento de software. El código fuente se debe reestructurar y alinear con una arquitectura de software actualizada, la cual siente las bases para su evolución como software, evitando así el problema inminente de construir otro sistema desde cero.

OBJETIVO

El objetivo general de este trabajo es diseñar una arquitectura de software para sistemas expertos e implementarla en el sistema experto de evaluación de riesgos naturales CAPRA versión 2.0.

Con esta nueva arquitectura para CAPRA, se pretenden lograr tres objetivos específicos:

- Establecer un marco de desarrollo de software para futuras versiones de este sistema experto, que permita su evolución hacia tecnologías de uso masivo como los dispositivos móviles y cómputo en la nube.
- Potenciar en CAPRA las características de modularidad, mantenibilidad, reutilización e incorporación de funcionalidad.
- Extender el ciclo de vida de CAPRA para contribuir a la ingeniería civil con una herramienta para la evaluación de riesgos naturales utilizable en planeación y diseño de obras.

A partir de la versión actual de CAPRA 2.0 se construirá una nueva versión del sistema experto como demostración de la implementación de la arquitectura de software diseñada en este trabajo.

CONTENIDO DEL TRABAJO

Este trabajo de tesis se compone de 6 capítulos con el siguiente contenido:

En el capítulo 1 se presentan los fundamentos teóricos de la evaluación de riesgos naturales sobre la cual se sustenta el dominio del sistema experto CAPRA; se describen los tres elementos de la evaluación del riesgo: amenaza, vulnerabilidad y exposición.

El capítulo 2 contiene una breve introducción a los sistemas expertos como parte de la inteligencia artificial, donde se justifica la razón de tratar a CAPRA como un sistema experto.

El capítulo 3 se centra en la arquitectura de software, se presentan las definiciones, fundamentos teóricos y la importancia que tiene el diseño de la arquitectura dentro del desarrollo de sistemas.

El capítulo 4 contiene la definición del proceso de diseño de la arquitectura utilizado, donde como resultado principal se documenta la nueva arquitectura de software mediante el modelo “4+1”, donde este último se compone de 5 vistas excluyentes generadas con diagramas utilizando el lenguaje de modelado unificado (UML, por sus siglas en inglés, Unified Modeling Language).

El capítulo 5 resume el resultado de los pasos ejecutados en la implementación de la arquitectura. Incluye la definición de un proceso propuesto para la reestructuración del

código en capas y entidades de negocio que integra al código de la versión actual de CAPRA en la nueva arquitectura.

El capítulo 6 integra las conclusiones de este trabajo y se describen los futuros desarrollos que podrían continuar a partir de este trabajo.

CAPÍTULO 1 EVALUACIÓN DE RIESGOS NATURALES

Los desastres naturales son eventos que marcan el destino económico de un país y sus habitantes. Los sismos, tsunamis, erupciones volcánicas, inundaciones y huracanes lamentablemente causan pérdidas irreparables de vidas humanas; además, ocasionan pérdidas materiales y económicas que ascienden a decenas de millones de dólares por evento.

Un sismo con epicentro en el océano puede producir un tsunami devastador. El tsunami de Banda Aceh en Indonesia del 26 de diciembre del 2004 ocasionó más de 227 mil pérdidas humanas en las islas de Indonesia (USGS, 2012).

Un sismo con epicentro cercano a una ciudad es aún más peligroso, aunque su impacto depende de la magnitud del mismo, y en mayor medida de la vulnerabilidad de las edificaciones. Por ejemplo, el terremoto del 12 de enero del 2010 con una magnitud de M_W=7 y epicentro a 15 km de Puerto Príncipe en Haití, le quitó la vida a más de 300 mil personas, destruyó 100 mil viviendas y dañó otras 200 mil. En consecuencia, hubo más de un millón de damnificados sin hogar y viviendo en campamentos (Benito, y otros, 2012).

Eventos naturales recientes han demostrado su alta capacidad de destrucción y el nivel de amenaza que representan. Cinco de los ocho sismos de mayor magnitud registrados en la historia (USGS, 2012) ocurrieron entre 1952 y 2012: M_W=9 en Kamchatka, Siberia (1952), M_W=9.5 en Valdivia, Chile (1960), M_W=9.2 en Anchorage, Alaska (1964), M_W=9.1 en Sumatra, Indonesia (2004) y M_W=9.0 en Honshu, Japón (2011).

Otros eventos no presentaron la mayor magnitud en su tipo pero al combinarse con características particulares de la región de impacto, también se convirtieron en catastróficos para la población. Dieciocho temblores con magnitud entre 6.6 a 7.9 grados en escala Richter ocurrieron entre 1915 y 2012; cada uno de ellos causó más de 20 mil muertes (USGS, 2012).

Un evento natural puede producir efectos devastadores sobre una ciudad, región o un país entero. Estos efectos se traducen en pérdidas de vidas humanas, pérdidas económicas directas e indirectas, endeudamiento y reducción del desarrollo socioeconómico.

Los huracanes Rita y Katrina se formaron en el Golfo de México durante septiembre y octubre del 2005. Ambos huracanes de categoría 5 en la escala Saffir-Simpson (SSHS) causaron pérdidas económicas por más de 108 mil millones de dólares (mdd) (CBO, 2005). El huracán Sandy formado durante octubre del 2012 con categoría SSHS=3 ocasionó pérdidas por más de 75 mil mdd.

En los dos ejemplos anteriores, el impacto económico superó al PIB de 117 países correspondiente al 2012, calculado por el Fondo Monetario Internacional, cuyos valores se situaron por debajo de 70 mil mdd. (FMI, 2013). Los desastres naturales afectan a todos y

cada país debe evaluar alternativas para crear una estrategia enfocada a la reducción de esos impactos.

Para mitigar y prevenir estas amenazas se requiere como primer paso cuantificar de manera razonable los riesgos a los que se expone una estructura, ciudad o región.

Este capítulo describe cómo calcular el riesgo al que se expone un elemento, de suerte que el resultado puede traducirse a indicadores puntuales de pérdidas económicas. Si bien el fundamento probabilístico de este modelo puede ser aplicado para cualquier tipo de amenaza, este capítulo se centra en la evaluación del riesgo por amenazas naturales.

EVALUACIÓN DEL RIESGO

La evaluación del riesgo consiste en modelar el daño que puede producir una amenaza natural sobre un elemento expuesto. El cálculo se realiza mediante un análisis probabilista donde se determina la distribución de probabilidades de las pérdidas que pueden sufrir los elementos como consecuencia de la ocurrencia de eventos amenazantes

El riesgo se cuantifica en términos de pérdidas, las cuales están en función del valor económico asociado para resarcir el daño causado al elemento evaluado. Este modelo de cálculo empleado permite determinar indicadores de riesgo en términos económicos y su cálculo se basa en tres componentes: La amenaza, los elementos expuestos y la vulnerabilidad.

El modelo de cálculo de pérdidas utilizado como fundamento teórico en este capítulo se basa en la Metodología de Evaluación Probabilista de Riesgos Naturales – CAPRA (ERN Consorcio, 2011).

COMPONENTES PARA LA EVALUACIÓN DEL RIESGO

El riesgo se determina a partir de tres componentes:

- La amenaza.- Se refiere al conjunto de eventos con potencial para producir daño y están representados por uno o varios parámetros que determinan la intensidad para un tipo específico de amenaza. En general, el valor de intensidad para un evento varía espacialmente.
- Los elementos expuestos.- Se refiere a la población, edificaciones, infraestructura u objetos susceptibles de sufrir un daño por una amenaza. El elemento es representado por una tipificación y atributos de la infraestructura además de la cantidad poblacional albergada, ubicación geográfica y valor económico.

- La vulnerabilidad.- Se refiere a la relación entre el posible daño causado a un tipo específico de infraestructura y la intensidad de la amenaza. Esa relación es representada por funciones de vulnerabilidad.

La amenaza y la vulnerabilidad son determinadas a partir de registros históricos, observaciones de campo después de un evento de intensidad conocida, ensayos de laboratorio y modelos analíticos sobre el comportamiento de componentes o estructuras.

La amenaza

La amenaza es cualquier tipo de evento que tiene potencial para causar daño sobre un elemento expuesto.

La capacidad destructiva de la amenaza se mide mediante al menos un parámetro de intensidad que, idealmente, esté muy fuertemente correlacionado con el daño presentado.

Algunos parámetros de intensidad para amenazas naturales se muestran en la tabla siguiente:

Amenaza	Efecto	Parámetro de Intensidad
Sismo	Movimiento del terreno	Aceleración, velocidad y desplazamiento máximos del terreno, y espectrales para diferentes periodos estructurales
Sismo	Tsunami	Profundidad máxima de inundación
Huracán	Vientos huracanados	Velocidades de viento pico para ráfagas de 3 segundos
Huracán	Marea de tormenta	Profundidad máxima del área de inundación
Huracán	Lluvia huracanada	Profundidad máxima de la inundación
Lluvias no huracanadas		Profundidad máxima de la inundación
Inundación		Profundidad y extensión del área de inundación
Deslizamientos		Distribución del factor de Inseguridad o indicador de susceptibilidad al deslizamiento
Erupción volcánica	Caída de cenizas	Distribución de espesores de ceniza
Erupción volcánica	Flujos de lava	Distribución del área de afectación
Erupción volcánica	Flujos piroclásticos	Distribución del área de afectación

Tabla 1-1 Parámetros de intensidad para amenazas naturales (ERN, 2011)

La caracterización de la amenaza es el primer componente en la evaluación del riesgo, y se representa por un conjunto de escenarios, donde cada escenario tiene asociado una distribución geográfica de intensidades específicas y una frecuencia de ocurrencia anual.

El cálculo de la intensidad de cada escenario tiene un nivel de incertidumbre originado por sus escasos datos de entrada y por la imposibilidad de saber dónde, cuándo y de qué intensidad serán los próximos eventos. En consecuencia, la intensidad se representa con una distribución de probabilidades con un valor esperado y un coeficiente de variación que represente a la incertidumbre.

La intensidad y la frecuencia de ocurrencia se determinan a partir de los registros históricos de ocurrencia de la amenaza. Cuando no se cuenta con medición instrumentada de la intensidad, entonces se recurre a la información histórica de efectos y pérdidas causadas, así como las características del evento reportado.

También se debe recurrir a instituciones o entidades relacionadas con el estudio de cada tipo de amenaza para recabar la información de los eventos presentados en la región.

Los elementos expuestos

Un fenómeno natural se convierte en un riesgo cuando existen elementos que pueden sufrir daño al momento de presentarse el evento.

En la evaluación del riesgo se agrupan los elementos expuestos en tres categorías: construcción, contenidos y población afectada.

Algunos ejemplos de construcciones son: puentes, túneles, puertos, presas, autopistas, aeropuertos, instalaciones de transporte colectivo, hospitales, escuelas, antenas de telecomunicación, líneas eléctricas, edificios para uso habitacional y de oficinas, instalaciones industriales, ductos de gas y petróleo, refinerías y termoeléctricas.

El daño a cada estructura se puede traducir en: pérdidas directas por el costo de reparar o reconstruir el bien; en pérdidas indirectas por el daño en los objetos y personas ubicadas dentro de la misma; y en pérdidas indirectas por lapso de inoperatividad de los servicios que presta. Estas últimas quedan fuera del alcance de la metodología de evaluación de riesgo descrita en el presente trabajo debido a que las líneas de interacciones muy complejas no son fáciles de modelar.

Para evaluar el riesgo de una estructura se requiere la siguiente información:

- Costo del elemento.- Se define como un valor económico aproximado para resarcir el daño ante un escenario de pérdida total. Este costo se considera al momento de convertir el porcentaje posible de daño a un valor de pérdida económica.
- Localización geográfica.- Se representa su ubicación mediante un punto, línea o polígono geo-referenciado. Esto permite representar las pérdidas utilizando mapas, así como saber la intensidad asignada en cada evento simulado.
- Cantidad de población dentro de la construcción.- Este valor se utiliza para determinar el posible número de pérdidas humanas o afectaciones en la población.

- Características constructivas.- Se utilizan para tipificar a la construcción y asociarla con la función de vulnerabilidad de dicho tipo constructivo.

Algunos ejemplos de características para infraestructura son: sistema estructural (marcos estructurales, losas planas), material predominante (concreto, mampostería, madera, concreto prefabricado), condiciones especiales (irregularidad, reforzamientos, columnas cortas, esbeltez excesiva, daños previos, hundimientos evidentes, edificación en esquina), fecha de construcción, material de pisos, tipo de cubierta (concreto, asbesto, lámina de acero, lámina de cartón).

El nivel de caracterización de los elementos depende del grado de detalle de la información recabada; la metodología estudiada propone tres: nivel ciudad o país, nivel región o zona homogéneas y nivel predio o edificación.

Para el caso de evaluación de riesgo a nivel de ciudad o país se puede recurrir a instituciones gubernamentales de estadística y censo para establecer valores promedio de habitantes por zona y definir áreas geográficas con infraestructura homogénea de tipo urbana, rural, residencial, comercial, industrial, de educación o salud.

Cuando se calcula el riesgo de una sola edificación, se pueden obtener las características detalladas mediante visita de campo, fotografías aéreas o mediante software como Google Earth y Google Maps.

La información de los elementos expuestos se almacena en bases de datos y archivos con formato especial para su procesamiento y representación gráfica utilizando software con soporte para Sistemas de Información Geográfica (GIS).

La vulnerabilidad

La vulnerabilidad es una medida que determina qué tan propensa es una estructura a sufrir daño ante un evento amenazante de una intensidad específica.

El daño que puede sufrir una estructura está determinado por sus características constructivas. Por ejemplo, un huracán categoría SSCS=5 induce mayor porcentaje de daño a una vivienda de madera que a un hospital de concreto y acero.

La vulnerabilidad de una construcción se mide como el porcentaje de daño esperado, mientras que la vulnerabilidad en la población se representa como el valor esperado de porcentaje de ocupantes afectados (heridos, desalojados, defunciones o el tipo de afectación que se defina) respecto al total de personas que alberga la estructura.

Función de vulnerabilidad

Los valores de vulnerabilidad para un tipo de estructura y para un tipo de amenaza en particular se expresan mediante una función de vulnerabilidad. Dichos valores se grafican

como una curva que relaciona el valor esperado del daño generado por cada una de las intensidades que presenta la amenaza.

El valor del daño no puede ser predicho exactamente, por tal razón es común suponer que el daño tiene una distribución de probabilidades beta que queda completamente definida con el valor esperado del daño y su desviación estándar.

En la figura 1-1 se muestra un ejemplo de una función de vulnerabilidad para estructuras de tipo auditorio o teatro, ante sismo. El eje vertical indica el porcentaje de daño esperado y la desviación estándar para cada uno de los valores de intensidad, mientras que el eje horizontal representa a la intensidad del sismo mediante el parámetro de aceleración máxima del suelo.

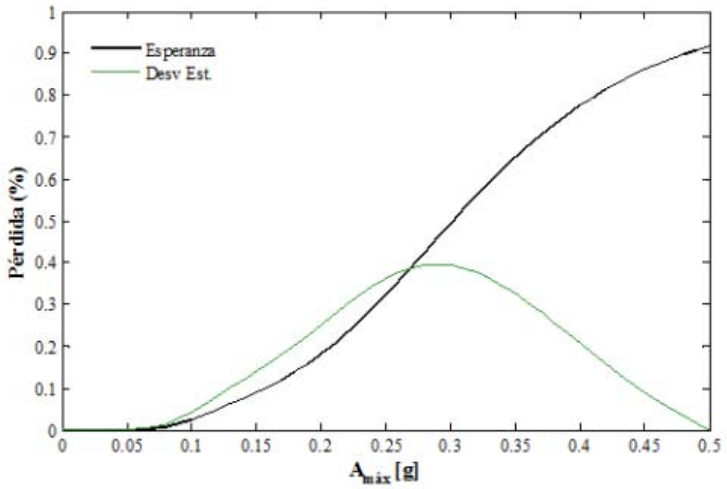


Figura 1-1 Función de vulnerabilidad para construcciones de tipo auditorio o teatro para diferentes intensidades de sismos (DOF, 2012)

Un evento geofísico puede generar más de un tipo de amenaza; por ejemplo un huracán trae consigo amenaza por viento, inundación, deslizamientos y marea de tormenta. Cada amenaza tendrá una función de vulnerabilidad distinta aun cuando se analice el mismo tipo de construcción.

ECUACIÓN BÁSICA DE CÁLCULO DEL RIESGO

El riesgo de una estructura está determinado por su valor económico, su función de vulnerabilidad, la intensidad específica de cada evento amenazante y la frecuencia de ocurrencia de cada uno de éstos. El modelo del cálculo se expresa con la ecuación básica 1.1, que es una forma del teorema de la probabilidad total:

$$v(p) = \sum_{i=1}^{Total\ de\ eventos} Pr\,(P > p|Evento\ i)\ \ F_A(Evento\ i)$$

Ec. 1.1

En esta ecuación el *Total de eventos* contabiliza todos los posibles eventos con potencial para causar daño, donde cada evento tiene una frecuencia anual de ocurrencia representada como $F_A(Evento\ i)$. En tanto que $Pr(P>p|Evento\ i)$ representa la probabilidad de que el daño que sufra el elemento expuesto supere el valor de *p* ante la ocurrencia del *i*-ésimo evento.

La suma de esos productos para el total de eventos se conoce como la *tasa de excedencia de la pérdida*, representada por *v(p)*, que indica la frecuencia anual con la cual será excedido un valor de pérdida específico representado por *p* (ERN Consorcio, 2011).

La figura 1-2 muestra dos ejemplos de curva de excedencia de pérdidas por sismo y huracán exclusivas para un conjunto de estructuras correspondientes a 4 Secretarías de Estado del gobierno de México.

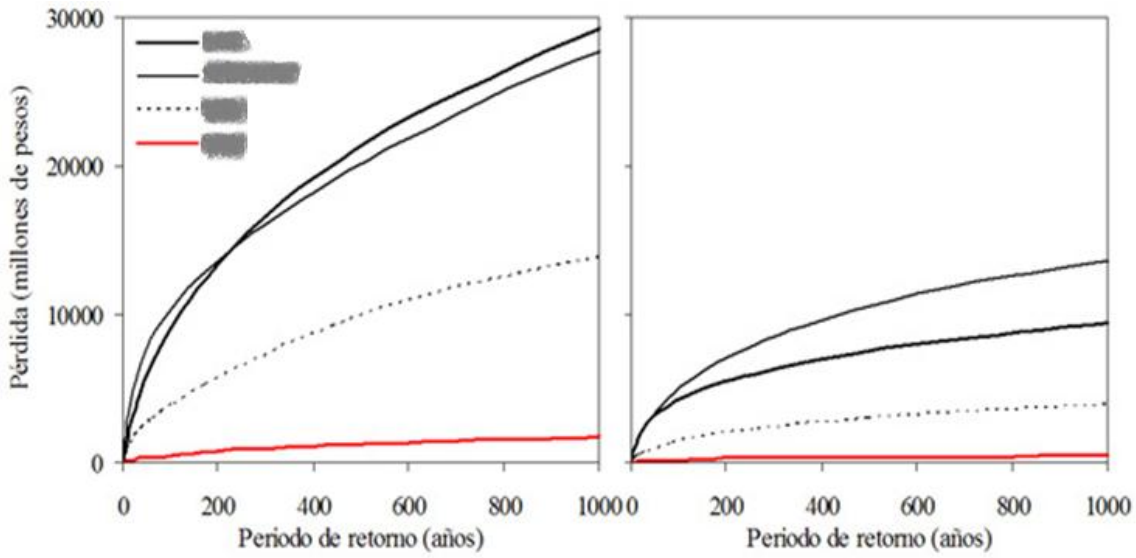


Figura 1-2 Curva de excedencia de pérdida para la infraestructura en México por eventos de Sismo (izquierda) y Huracán (derecha). Los nombres de las secretarias se taparon intencionalmente. (Reinoso, Jaimes, Ordaz, & Niño, 2010)

Indicadores puntuales del riesgo

Si bien la curva de excedencia de pérdida (figura 1-2) sintetiza la ocurrencia de eventos que producen cierto nivel de pérdidas, en ocasiones se hace necesario caracterizar la pérdida con un único valor; para ello se utilizan los indicadores puntuales del riesgo siguientes:

La Pérdida Anual Esperada (PAE) se obtiene con la sumatoria de los resultados de multiplicar la pérdida esperada de cada uno de los eventos que conforman la amenaza por su probabilidad de ocurrencia en el lapso de un año:

$$PAE = \sum_{i=1}^{Total\ de\ eventos} E\,(P|Evento\ i)\ F_A(Evento\ i)$$

Ec.1.2

La Pérdida Máxima Probable (PML, por Probable Maximum Loss) indica el valor de la pérdida que puede tener una estructura ante un evento de gran intensidad pero de baja frecuencia. En la industria de seguros se calcula el PML como la pérdida asociada a periodos de retorno de entre 200 y 1500 años. En cada tipo de amenaza se debe asegurar que el valor del periodo de retorno incluya al evento de mayor intensidad ocurrido en la región.

La Prima Pura de Riesgo (PPR) es un indicador utilizado por la industria aseguradora para asignar un valor a la póliza de riesgo de una construcción. Este indicador se calcula al dividir el valor de la PAE entre el costo del bien asegurado.

Incertidumbre

Debido a que la ocurrencia de un evento catastrófico, su intensidad y el daño que ocasiona son impredecibles, entonces es indispensable introducir un factor de incertidumbre. Adicionalmente, existen otros niveles de incertidumbre asociados a la calidad de la información de los elementos expuestos, la determinación de la vulnerabilidad y la imposibilidad de un cálculo exacto de las pérdidas; estos valores se agrupan en un nivel global de incertidumbre asociado a los valores finales de pérdida calculados.

En términos generales, la probabilidad de que la pérdida supere un valor dado durante un evento no puede calcularse de manera directa. Es costumbre entonces calcularla "encadenando" las siguientes dos distribuciones de probabilidad:

$$Pr(P > p|Evento) = \int_I Pr(P > p|I) f(I|Evento) dI$$

Ec. 1.3

El término $Pr\ (P > p\ |I)$ es la probabilidad de que un valor de pérdida p sea superado dado que ocurrió un evento de intensidad I; esta probabilidad puede calcularse empleando las funciones de vulnerabilidad. Por otra parte, la incertidumbre en la intensidad se incorpora en la ecuación 1.3 a través del término $f\ (I\ |Evento)$, el cual representa la densidad de probabilidades de la intensidad, dada la ocurrencia del evento en cuestión.

CAPÍTULO 2 SISTEMAS EXPERTOS

La forma de programar sistemas expertos ha evolucionado notablemente. Durante la década de los 90 un programa creado en LISP (acrónimo de List Processor) o PROLOG (acrónimo de PROgrammation en LOGique) con menos de cien reglas de conocimiento obtenidas de un especialista era considerado un sistema experto; actualmente los sistemas empresariales y el software especializado logran tamaños de hasta cientos de miles de líneas de código.

Este capítulo resume la definición, características y lenguajes de programación de los sistemas expertos, y establece sus diferencias con los sistemas especializados.

Al final, se presenta al sistema experto que será trabajado en los siguientes capítulos como proceso de demostración central en este documento de tesis.

SISTEMAS EXPERTOS

Los sistemas expertos son parte de la solución que ofrece la Inteligencia Artificial (IA) para la representación del conocimiento y su utilización. La inteligencia artificial es un área de las ciencias de la computación cuyo objetivo es abstraer los procesos mentales que constituyen la inteligencia humana e implementarlos mediante elementos de cómputo.

Por su parte, Frenzel (1989) sostiene que un sistema experto es un tipo especial de software que incorpora el razonamiento y la información que posee al menos un especialista sobre un campo específico de conocimiento. Su objetivo principal es entregar el conocimiento del especialista a los usuarios para ayudarlos en sus actividades.

El mismo autor clasifica dicho conocimiento en:

- Conocimiento declarativo o descriptivo. Es el conjunto de afirmaciones (hechos) con las que se describe a los conceptos y sus relaciones. Son las fragmentos de conocimiento que describen el qué, quién y dónde.
- Conocimiento procedimental o prescriptivo. Son las instrucciones de cómo aplicar el conocimiento declarativo y representan el razonamiento de cómo, cuándo y/o por qué se realiza una determinada acción.

Características

La primera generación de sistemas expertos estaba diseñada para incorporar a la computadora el conocimiento mediante reglas de producción y la solución de problemas se obtenía a través de un motor de inferencia.

Las reglas de producción se almacenan en una base de conocimiento. Estas reglas se representan gráficamente como árboles de hechos o decisiones. Por otra parte, el motor de inferencia aplica diferentes técnicas de búsqueda y comparación sobre la base de conocimiento hasta encontrar una respuesta.

Adicionalmente, estos programas de cómputo contaban con un subsistema o módulo para adquisición de nuevas reglas, una interfaz gráfica para interactuar con el usuario y una base de datos para registrar información temporal derivada de la ejecución del sistema. La figura 2-1 muestra una arquitectura común para esa primera generación de sistemas expertos.

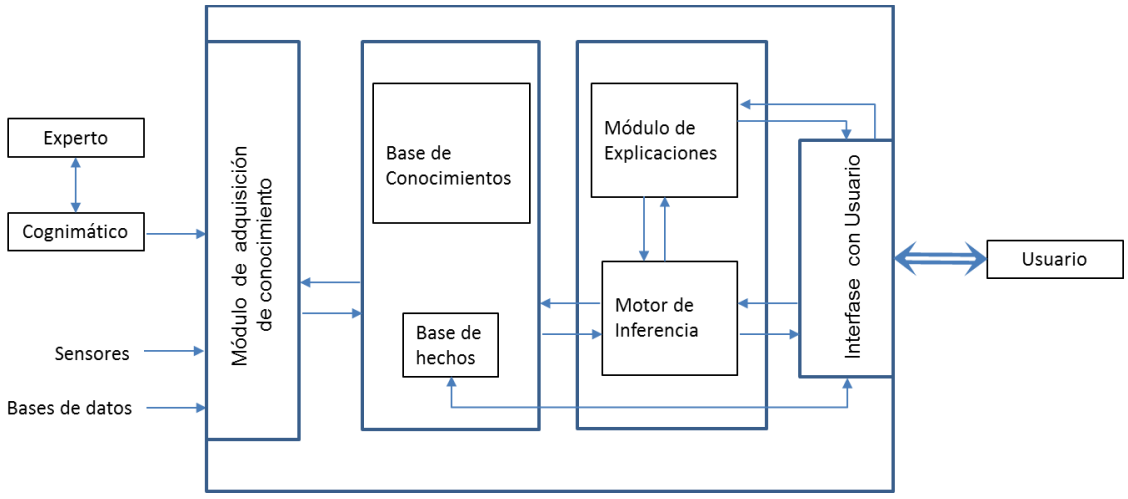


Figura 2-1 Arquitectura de un Sistema Experto (Chatain & Dussauchoy, 1988)

Lenguajes de programación

Desde el nacimiento de la Inteligencia Artificial en 1950 y hasta 1995, los investigadores y programadores usaban equipos de cómputo con recursos limitados. La supercomputadora de esa época tenía 500MB en memoria de acceso aleatorio (RAM), 1 procesador de 3Ghz y disco duro de 1GB (Perez, 2008).

Esas limitantes influían en los lenguajes de programación. Por un lado, se necesitaba que el código fuente y los datos ocuparan el mínimo de bytes para almacenamiento; por el otro, se necesitaban algoritmos de alto desempeño que optimizaran el uso del espacio en memoria RAM y la capacidad del procesador.

En el caso de los lenguajes expertos el reto también implicaba almacenar el conocimiento en el menor espacio posible y desarrollar algoritmos capaces de procesar e interpretar ese conocimiento.

Los lenguajes LISP y PROLOG cubrían esas necesidades y estaban construidos para soportar totalmente la programación lógica y declarativa de reglas de producción e incluir un motor de inferencias propio.

Por dichas razones, ambos lenguajes tuvieron un auge en el desarrollo de sistemas expertos. Una veintena de dialectos de LISP fueron publicados por diferentes universidades y empresas.

Tal variedad de versiones trajo consigo la incompatibilidad del código fuente al momento de reutilizarlo. El problema de incompatibilidad para LISP se resolvió en 1994 con el estándar ANSI X3.226-1994 implementado en la versión Common Lisp y para PROLOG en 1995 con el estándar ISO/IEC 13211-1 conocido como ISO-PROLOG.

A pesar de las limitaciones en los equipos de cómputo, se podían desarrollar sistemas expertos en cualquier lenguaje de programación (Bauer, y otros, 1988).

Por otro lado, el paradigma orientado a objetos inició con los lenguajes Smalltalk y Simula67. Sin embargo, este paradigma tomó fuerza y se popularizó a partir de 1995 con el lanzamiento del lenguaje Java versión 1.1 realizado por la empresa Sun Microsystems. Siete años después, Microsoft lanzó la plataforma de desarrollo dotNet con tres lenguajes orientados a objetos: C#, VB.net y F#.

El paradigma orientado a objetos permite organizar el conocimiento aun cuando éste se exprese en términos de funciones, reglas, lógica, redes neuronales o lenguajes relacionales. De igual manera, permite interconectar diferentes tecnologías de software (Figura 2-2).

Figura 2-2 La orientación a objetos se puede utilizar como un mecanismo que organice e integre los diversos enfoques de sistemas. (Martin & Odell, 1997)

Al momento de redactar este trabajo, año 2014, el paradigma orientado a objetos goza de un amplio dominio en el desarrollo de software y una gran variedad de herramientas, arquitecturas, patrones de diseño y procesos de desarrollo se han creado exclusivamente para este paradigma. En cuanto a poder de cómputo, las actuales supercomputadoras son clúster (conjunto de computadoras sobre un mismo hardware que trabajan como un solo equipo de cómputo) que rebasan los 100TB en memoria RAM y 180mil procesadores que juntos logran 1,700 billones de operaciones de punto flotante por segundo (1.7 petaflops) (Perez, 2008).

La alta capacidad de cómputo permite que los lenguajes OO sean una opción para el desarrollo de la nueva generación de sistemas expertos. Incluso, en estos lenguajes se puede incorporar componentes con motores de inferencia desarrollados en LISP y PROLOG.

Sistemas expertos vs sistemas especializados

Durante la primera generación de sistemas expertos, el conocimiento de toda disciplina era un aspirante a convertirse en un sistema experto: bastaba con que se lograra documentar su conocimiento. Sin embargo, actualmente todos los sistemas cuentan con cierto grado de conocimiento especializado de una o varias disciplinas.

Por esta razón, existe una delgada línea entre los sistemas expertos y los sistemas especializados. Los primeros, implementan el conocimiento de una o varias disciplinas de las ciencias y/o ingenierías; mientras que, en los sistemas especializados, el conocimiento proviene de los procedimientos funcionales que poseen los usuarios de negocio y/o empresas.

SISTEMA EXPERTO PARA LA EVALUACIÓN DE RIESGOS NATURALES

La metodología de evaluación de riesgos naturales descrita en el capítulo anterior está implementada en el sistema experto CAPRA. Este software forma parte de la Plataforma de Información sobre Riesgos de Desastres publicada en internet bajo el dominio www.ecapra.org.

La plataforma comenzó en enero de 2008 como una asociación entre el Centro de Coordinación para la Prevención de Desastres Naturales en Centroamérica (CEPREDENAC), la Estrategia Internacional para la Reducción de Desastres de la ONU, el Banco Interamericano de Desarrollo (BID) y el Grupo de Gestión del Riesgo de Desastres del Banco Mundial para América Latina y el Caribe.

Su portal web define a CAPRA como una plataforma código abierto (eCAPRA.org, 2012), por lo que sus programas ejecutables y el código fuente pueden ser descargados libremente desde su dirección en internet. Estos programas están desarrollados con el lenguaje orientado a objetos VB.Net.

El sistema experto CAPRA incluye módulos para el cálculo de amenazas, exposición, vulnerabilidad sísmica, daños y pérdidas, y ha sido utilizado en países de América Latina y sur de Asia para la planeación en la prevención de desastres naturales y en el diseño de estrategias de gestión de riesgos.

Nuevas necesidades y retos

En 1989, Luis Frenzel propuso diez pasos básicos para la creación de un sistema experto (Frenzel). En menos de veinte años, esos pasos han sido reemplazados por procesos, metodologías y modelos de desarrollo de software de mayor complejidad.

El primer reto al crear sistemas expertos debería ser encapsular el conocimiento del especialista en programas de cómputo. Sin embargo, el software ha evolucionado de tal manera, que tanto los patrocinadores de un desarrollo como los usuarios exigen nuevas necesidades, entre ellas:

> Productividad en el código: se requiere una codificación acelerada, con alta calidad y que permita integrar nueva funcionalidad en menos tiempo y con el menor impacto.
>
> Omnipresencia del sistema: se espera que el sistema esté disponible para el usuario desde cualquier lugar, dispositivo y en todo momento. Esto se traduce en requerimientos de alta disponibilidad, ejecución multiplataforma y movilidad.
>
> Respuesta en tiempo real: el usuario necesita una solución y/o resultados de forma inmediata al presentarse el problema. Esto demanda procesamiento paralelo, escalabilidad y redundancia.

Los investigadores y expertos, sin patrocinio suficiente para tener a un grupo de ingenieros de software, se pueden enfrentar con limitaciones para cumplir con esas características.

En el capítulo siguiente se aborda el tema de la arquitectura de software y después se propone el diseño una arquitectura simple que cubre parte de esos retos y necesidades.

Con esta nueva arquitectura se pretende que el investigador o especialista se centre principalmente en la programación de componentes que representen el conocimiento sobre evaluación de riesgos naturales. Estos componentes forman parte de la nueva versión el sistema experto CAPRA la cual es resultado de la implementación de dicha arquitectura, la cual se describe en el último capítulo del presente trabajo.

CAPÍTULO 3 ARQUITECTURA DE SOFTWARE

Este capítulo presenta una introducción a la arquitectura de software, su definición y alcance, así como los tipos básicos de arquitecturas, estilos arquitectónicos y ejemplos arquetipos.

También se destaca la importancia del diseño de la arquitectura dentro del desarrollo de sistemas y se enuncian las tres características principales que debe cubrir el diseño de la arquitectura.

ARQUITECTURA DE SOFTWARE

La construcción exitosa de un sistema requiere de un proceso de desarrollo de software; sin embargo cualquier construcción está destinada al fracaso si no tiene un diseño adecuado y completo.

En el campo de la ingeniería civil, el problema del diseño inadecuado o inexistente es evidente cuando un sismo de mediana intensidad destruye edificaciones ocasionando enormes pérdidas económicas a propietarios y gobiernos.

En el desarrollo de sistemas, el diseño es una etapa indispensable con afectación directa en el éxito o fracaso de un proyecto. El diseño adecuado y completo de los sistemas de cómputo es tratado por la disciplina llamada arquitectura de software.

La arquitectura de software forma parte de la ingeniería software. La arquitectura de un sistema es el resultado del diseño de alto nivel de los componentes de software que forman parte de la solución tecnológica de un problema o necesidad.

Cabe mencionar que existen más de una docena de enfoques o escuelas sobre la arquitectura de software. Incluso, entre ellas existen controversias sobre su terminología, alcance y definición, tal como lo describen Malveau y Mowbray (2004).

Por tal razón es que existen varias definiciones sobre arquitectura de software. A continuación se mencionan algunas.

Bass, Clements y Kazman (2003) definen: “La arquitectura de software de un programa o sistema de cómputo es la estructura o estructuras del sistema, que comprenden elementos de software, las propiedades externamente visibles de esos elementos, y la relación entre ellos.”.

Por su parte, Clements (2010), la describe como "El conjunto de las estructuras necesarias para razonar acerca del sistema, que comprende elementos de software, las relaciones entre ellos, y las propiedades de ambos."

Taylor, Medvidovic y Dashofy (2010) indican que "La arquitectura de software es el conjunto de las principales decisiones de diseño que gobiernan un sistema.".

Mientras que el Software Engineering Institute (SEI) (2014) con formalidad define: "La arquitectura de software de un programa o sistema de cómputo es una representación del sistema que ayuda en la comprensión de cómo éste se comportará. Arquitectura de software sirve como modelo para el sistema y para el desarrollo del proyecto. La arquitectura es el portador primario de las cualidades del sistema, tales como el rendimiento, la modificabilidad y la seguridad, ninguno de los cuales se pueden alcanzar sin una visión arquitectónica unificadora. La arquitectura es un artefacto para el análisis temprano para asegurarse de que un enfoque de diseño dará lugar a un sistema aceptable."

La arquitectura de software, como disciplina, ha evolucionado aceleradamente. Su campo de acción crece con la misma tendencia con que se publican nuevas versiones de procesos de desarrollo de software, lenguajes de programación y frameworks para generar código.

Lo anterior ha originado que la propia definición de arquitectura de software siga en evolución. Algunas de las definiciones pueden parecer cortas, aunque en realidad están bien acotadas.

Por otra parte, en el campo laboral, los objetivos y expectativas de esta disciplina se suelen distorsionar por dos factores: la confusión con la arquitectura empresarial y las actividades del arquitecto de software

Arquitectura empresarial

Cuando se construye un sistema de cómputo empresarial se presentan retos tecnológicos, cuyas soluciones son parte de la arquitectura del sistema. Algunos ejemplos de ello son: crear una infraestructura de servidores como un data center, definir una red privada entre oficinas ubicadas en diferentes países, seleccionar un manejador de base de datos comercial con capacidad en peta bytes o evaluar proveedores de servicios en la nube.

En dichos ejemplos, el líder o dueño del proyecto comúnmente espera que el arquitecto de software proponga soluciones a dichos retos tecnológicos debido a que considera que son parte del alcance de la arquitectura. Tal consideración proviene de tergiversar las definiciones de arquitectura de software y arquitectura empresarial.

La arquitectura de software es un componente dentro de la arquitectura empresarial, por lo que ésta última tiene un mayor alcance.

El consorcio global The Open Group publicó TOGAF (por las siglas inglés de The Open Group Architecture Framework) como un marco de trabajo para la arquitectura empresarial, la cual se conforma de la siguiente manera.

- Arquitectura de aplicaciones. Enfocada a la arquitectura de software.
- Arquitectura tecnológica. Referente al hardware y redes.
- Arquitectura de datos. Gestiona los datos e información física o lógica.
- Arquitectura de negocios. Referente a procesos de negocio y flujos operativos.
- Arquitectura empresarial. Coordina a todas las anteriores para establecer políticas de gobernabilidad, gestión de datos, procesos y estrategias a nivel organizacional.

La separación de la solución tecnológica en arquitecturas distintas pero complementarias, permite acotar más claramente los objetivos y alcance de la arquitectura de software.

Actividades del arquitecto de software

Otro factor que suele distorsionar la expectativa de la arquitectura de software es el propio rol del arquitecto de software. El arquitecto de software resuelve diversas problemáticas durante la ejecución de un proyecto, las cuales pueden ser tanto técnicas como administrativas.

Ante una problemática, el arquitecto debe aplicar conocimientos tanto de arquitectura de software como de la ingeniería de software. Esto se debe a que la solución implica un mayor campo de acción y otras áreas de conocimiento, tal como lo define Sommerville (2004):

“La ingeniería de software se preocupa por cubrir todos los aspectos de la producción de software desde las fases tempranas de definición del sistema hasta el mantenimiento después de que éste ha sido desplegado en producción”.

Como reflejo de las soluciones ofrecidas por el arquitecto de software, se suele creer que el conocimiento aplicado corresponde únicamente a su disciplina y con ello le atribuyen otros objetivos que en realidad corresponden a la ingeniería software.

Otro ejemplo similar se presenta cuando los ingenieros de software suelen ejecutar más de un rol. En algunos casos realizan las actividades propias del arquitecto de software más las actividades correspondientes al líder técnico, pero con esa mezcla de actividades se genera una falsa creencia de que la arquitectura de software también lleva la consigna de la gestión de recursos humanos, materiales y económicos.

Si bien es cierto que tanto el líder técnico como el arquitecto de software deben tener un gran dominio de la tecnología, en la práctica solo existe un subconjunto de actividades que pueden realizar cualquiera de las dos posiciones; el resto son excluyentes.

Las actividades principales comunes y excluyentes de ambos roles se presentan en la figura siguiente.

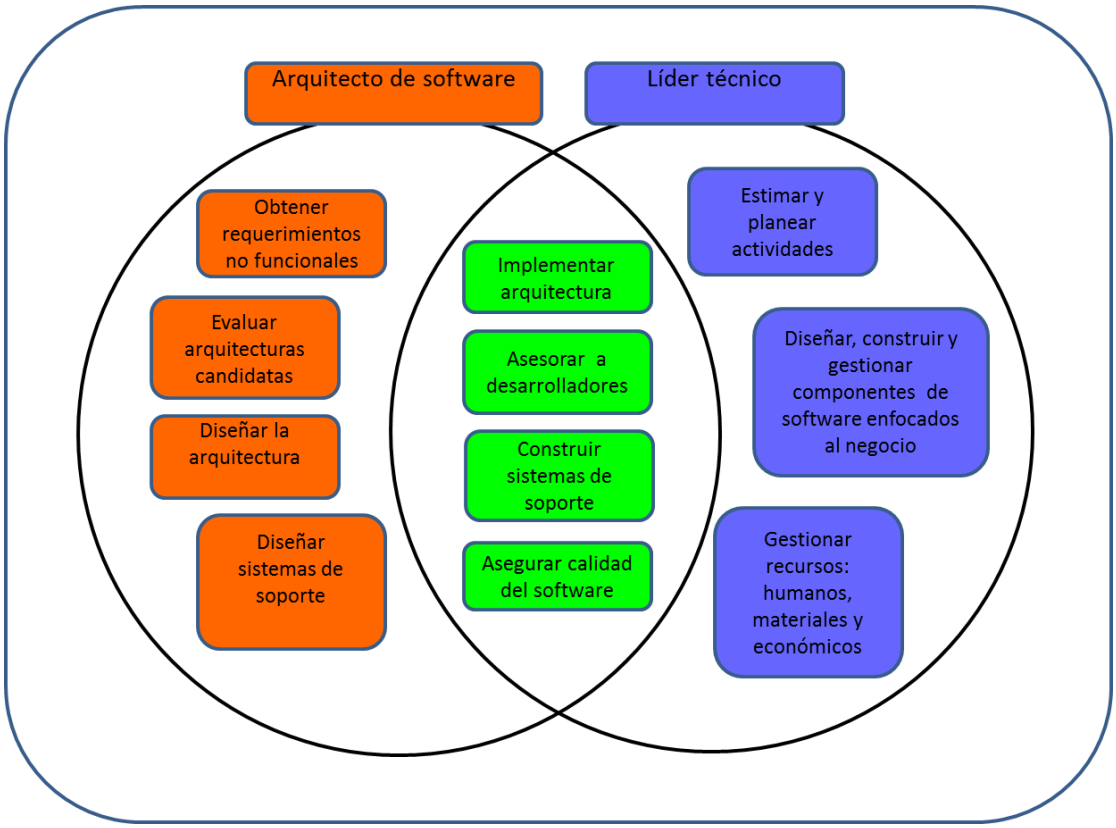


Figura 3-1 Diagrama con las actividades principales comunes y excluyentes entre el arquitecto de software y el líder técnico (Elaboración propia, 2014).

Los objetivos y expectativas de la arquitectura de software quedan cubiertos por las actividades del arquitecto de software y el cumplimiento de ellas repercute fuertemente en las fases restantes del desarrollo del sistema.

LA IMPORTANCIA DE LA ARQUITECTURA DE SOFTWARE

La arquitectura de software es esencial en la construcción de un sistema porque ésta debe cubrir los requerimientos funcionales y no funcionales, las necesidades del negocio, ofrecer una vida útil del software lo suficientemente larga para exceder el periodo de retorno de la inversión, además de contribuir en la productividad de los usuarios.

En el libro “Microsoft application architecture guide” (Microsoft patterns & practices, 2009) se advierte sobre la importancia de la arquitectura de la siguiente manera:

"Los riesgos expuestos por una arquitectura pobre incluyen software que es inestable, incapaz de soportar requerimientos de negocio existentes o futuros, o difícil de instalar o administrar en un ambiente de producción".

La importancia del diseño de la arquitectura de software también se ve reflejado en beneficios como lo enuncian Bass et al. (2003):

- *Exigencia sobre la fase de análisis del sistema, debido a que las decisiones de diseño de la arquitectura requieren información de los requerimientos críticos del sistema a cubrir como desempeño, confiabilidad y mantenibilidad.*
- *Reutilización de componentes para sistemas similares a través los modelos de la arquitectura que describen cómo se organizan e interactúan los componentes o subsistemas.*
- *Mejora en la comunicación del arquitecto de software con los agentes involucrados (stakeholders) mediante documentación gráfica del sistema de alto nivel para su entendimiento y discusión.*

Por su lado, Taylor define la arquitectura de software como el conjunto de las principales decisiones de diseño que gobiernan un sistema.

Estas decisiones de diseño son de suma importancia justamente porque repercuten sobre aspectos del sistema como: su estructura, comportamiento, interacción, implementación, calidad, seguridad, desempeño, usabilidad, experiencia de usuario, resiliencia, escalabilidad, despliegue, mantenibilidad e impacto económico para su desarrollo y mantenimiento.

Debido al gran impacto de tales decisiones, es que Taylor, Medvidovic y Dashofy (2010) califican la importancia de la arquitectura de software con la frase siguiente:

> ***"La arquitectura de software debe ser el corazón mismo del diseño y desarrollo de sistemas de software."***

ARQUITECTURA DE UNA APLICACIÓN

La arquitectura de software de un sistema de cómputo se define por tres características principales:

- Tipo de aplicación. Esta característica define las cualidades generales de la interface de la aplicación y si ésta se ejecuta total o parcialmente en un equipo, dispositivo, o con mediante un intermediario como un navegador web o plugin.
- Estilo arquitectónico. Esta característica define un aspecto referente a la comunicación, despliegue, dominio o estructura. A su vez, un tipo de aplicación estará conformado por uno o más estilos arquitectónicos.
- El arquetipo de la aplicación. Es una combinación de estilos arquitectónicos que conforman una solución común o recurrente para un tipo de aplicación.

Esas tres características se identifican, seleccionan y especifican como parte del diseño de la arquitectura de la aplicación.

En las siguientes secciones se presentan ejemplos de estas características, que pretenden ser agnósticas a la tecnología, a excepción de los arquetipos.

Es pertinente mencionar que los arquetipos referenciados en este trabajo están asociados a tecnologías de Microsoft y forman parte del libro “Guía de arquitectura N-Capas orientadas al dominio con .Net 4.0” (de la Torre, Zorrilla, Calvarro, & Ramos, 2010)

Se eligieron estos arquetipos debido a que el sistema experto CAPRA está construido con Visual Basic.Net. Y dicho lenguaje de programación forma parte de la plataforma de desarrollo de Microsoft con lo cual se garantiza la compatibilidad y reutilización del código actual del software.

TIPOS DE APLICACIONES

De la Torre (2010) describe 5 tipos básicos de aplicaciones, a partir de los cuales se puede crear un amplio espectro de soluciones híbridas.

1. Aplicaciones de Escritorio (Rich/Smart Client Applications)
2. Aplicaciones Web (Web Applications)
3. Aplicaciones de servicios (Service Applications)
4. Aplicaciones RIA (Rich Internet Applications)
5. Aplicaciones Móviles (Mobile Applications)

Los 5 tipos básicos derivan de la arquitectura física cliente/servidor, donde el “cliente” se representa por un equipo de cómputo, dispositivo, navegador web o componente plugin; en

tanto el "servidor" se representa como uno o varios servidores físicamente distribuidos o centralizados (cluster).

Adicionalmente, De la Torre publica la siguiente tabla con el resumen de las ventajas que ofrece cada tipo y las consideraciones de relevancia al momento de seleccionar el tipo de aplicación para un sistema.

Tipo de aplicación	Ventajas	Consideraciones
Aplicaciones para dispositivos móviles	• Sirven en escenarios sin conexión o con conexión limitada. • Se pueden llevar en dispositivos de mano. • Ofrecen alta disponibilidad y fácil acceso a los usuarios fuera de su entorno habitual.	• Limitaciones a la hora de interactuar con la aplicación. • Tamaño de la pantalla reducido.
Aplicaciones de escritorio	• Aprovechan mejor los recursos de los clientes. • Ofrecen la mejor respuesta a la interacción, una interfaz más potente y mejor experiencia de usuario. • Proporcionan una interacción muy dinámica. • Soportan escenarios desconectados o con conexión limitada.	• Despliegue complejo. • Versionado complicado. • Poca interoperabilidad.
RIA (Rich Internet Applications)	• Proporcionan la misma potencia gráfica que las aplicaciones de escritorio. • Ofrecen soporte para visualizar contenido multimedia. • Despliegue y distribución simples.	• Algo más pesadas que las aplicaciones web. • Aprovechan peor los recursos que las aplicaciones de escritorio. • Requieren tener instalado un plugin para funcionar.
Aplicaciones orientadas a servicios	• Proporcionan una interfaz muy desacoplada entre cliente y servidor. • Pueden ser consumidas por varias aplicaciones sin relación. • Son altamente interoperables.	• No tienen interfaz gráfica. • Necesitan conexión a internet.
Aplicaciones web	• Llegan a todo tipo de usuarios y tienen una interfaz de usuario estándar y multiplataforma. • Son fáciles de desplegar y de actualizar.	• Dependen de la conectividad a red. • No pueden ofrecer interfaces de usuario complejas.

Tabla 3-1. Ventajas y consideraciones tipos de aplicaciones. (de la Torre, Zorrilla, Calvarro, & Ramos, 2010)

Este resumen de ventajas y consideraciones son insumo para realizar una selección acertada del tipo de aplicación a construir.

Selección del tipo de aplicación

Para diseñar una arquitectura, el primer paso es seleccionar el tipo de aplicación. En ocasiones se requiere una solución tecnológica híbrida que incluye más de una aplicación, cada una con diferente tipo de arquitectura.

Una actividad del arquitecto de software es evaluar la mejor alternativa de solución a partir de los requerimientos no funcionales y de las restricciones de negocio, financieras y tecnológicas del proyecto. Dichas restricciones pueden ser variadas y con pesos diferentes, lo cual puede implicar una evaluación compleja.

Comúnmente el tipo de aplicación está sujeto a las limitaciones de instalación en los clientes, y las restricciones del sistema operativo sobre la ejecución de la aplicación.

En la figura siguiente se presenta un cuadrante con las alternativas para seleccionar el tipo de aplicación de dicho escenario básico, donde sólo intervienen dos restricciones: la homogeneidad de los sistemas operativos y la restricción de instalar software en los clientes.

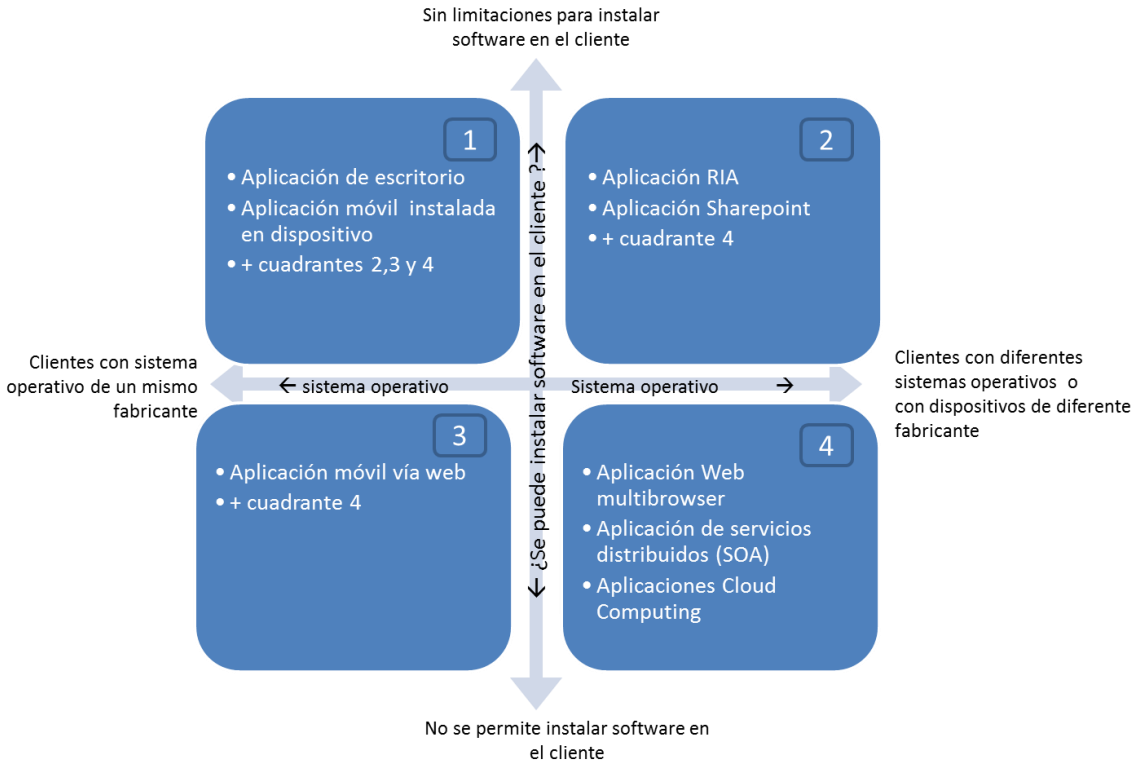


Figura 3-2. Cuadrante de decisión para seleccionar un tipo de arquitectura (Elaboración propia, 2014)

Por ejemplo, cuando no se permite instalar software en el cliente o existe incertidumbre de lograr una instalación exitosa, y los clientes tienen diversidad de sistemas operativos entonces se debe optar por una aplicación web, la cual estará restringida por las limitantes de seguridad y compatibilidad del navegador web del cliente (ver cuadrante 4).

En el caso opuesto, cuando se está permitido instalar software en los equipos de cómputo del cliente y además tienen el mismo sistema operativo, entonces se podrá explotar los recursos de los equipos como procesamiento en paralelo, almacenamiento local, acceso a periféricos. En consecuencia, se podrá seleccionar una aplicación de escritorio o móvil (ver cuadrante 1) y por ende se podrá ejecutar el mayor código posible en el cliente para reducir o anular la demanda de procesamiento en los servidores.

ESTILOS ARQUITECTÓNICOS

La segunda característica de que debe cubrir la arquitectura de un sistema son los estilos arquitectónicos que detallarán al tipo de aplicación seleccionado.

En este sentido, Microsoft patterns & practices (2009) define así a los estilos arquitectónicos:

"Un conjunto de principios de un patrón de grano grueso (patrón de nivel de aplicación) que proporciona un marco abstracto para una familia de sistemas. Un estilo arquitectónico mejora la partición y promueve la reutilización del diseño, aportando soluciones a los problemas más frecuentes que se repiten."

Los estilos arquitecturales se muestran en la siguiente tabla y se describen a detalle en el capítulo 3 de la misma fuente.

Aspecto	Estilos arquitecturales
Comunicaciones	Orientada a Servicios (SOA), Message Bus, Tuberías y filtros.
Despliegue	Cliente/Servidor, 3-Niveles (3-Tier), N-Niveles (N-Tier).
Dominio	Modelo de dominio (Model Driven Design).
Estructura	Componentes, Orientado a Objetos, Arquitectura en capas.

Tabla 3-2 Estilos arquitecturales (Microsoft patterns & practices, 2009)

De la Torre (2010) nombra a los estilos arquitectónicos como estilos estructurales y los describe así:

"Los estilos estructurales son *patrones* a nivel de aplicación que definen un aspecto del sistema que estamos diseñando y representan una forma estándar de definir o implementar dicho aspecto."

Como se observa en la Tabla 3-2 Estilos arquitecturales cada aspecto tiene más de un estilo arquitectónico que lo cubre, por tanto la selección de éstos debe basarse en las ventajas que ofrece cada uno para satisfacer los requerimientos no funcionales particulares del sistema a construir.

ARQUETIPOS DE APLICACIÓN

La tercera característica que define a la arquitectura de software de un sistema es el conjunto de componentes de software que conforman su estructura lógica. Esta estructura se modela con una vista lógica, en la cual los estilos estructurales seleccionados se detallan a nivel de componentes.

Dicha estructura lógica de componentes puede derivarse de propuesta de arquitectura de software genérica llamada arquetipo.

Al respecto, De la Torre (2010) propone y describe a detalle 8 arquetipos para los tipos de aplicación más comunes.

- Arquetipo Aplicación Web
- Arquetipo Aplicación RIA
- Arquetipo Aplicación Rica (Rich/Smart Client)
- Arquetipo Servicio Distribuido (SOA)
- Arquetipo Aplicación Rica Móvil
- Arquetipo Aplicaciones cloud computing
- Arquetipo Aplicaciones OBA.
- Arquetipo Aplicaciones de negocio basada en SharePoint

A manera de ejemplo, a continuación se muestra el arquetipo para aplicación web.

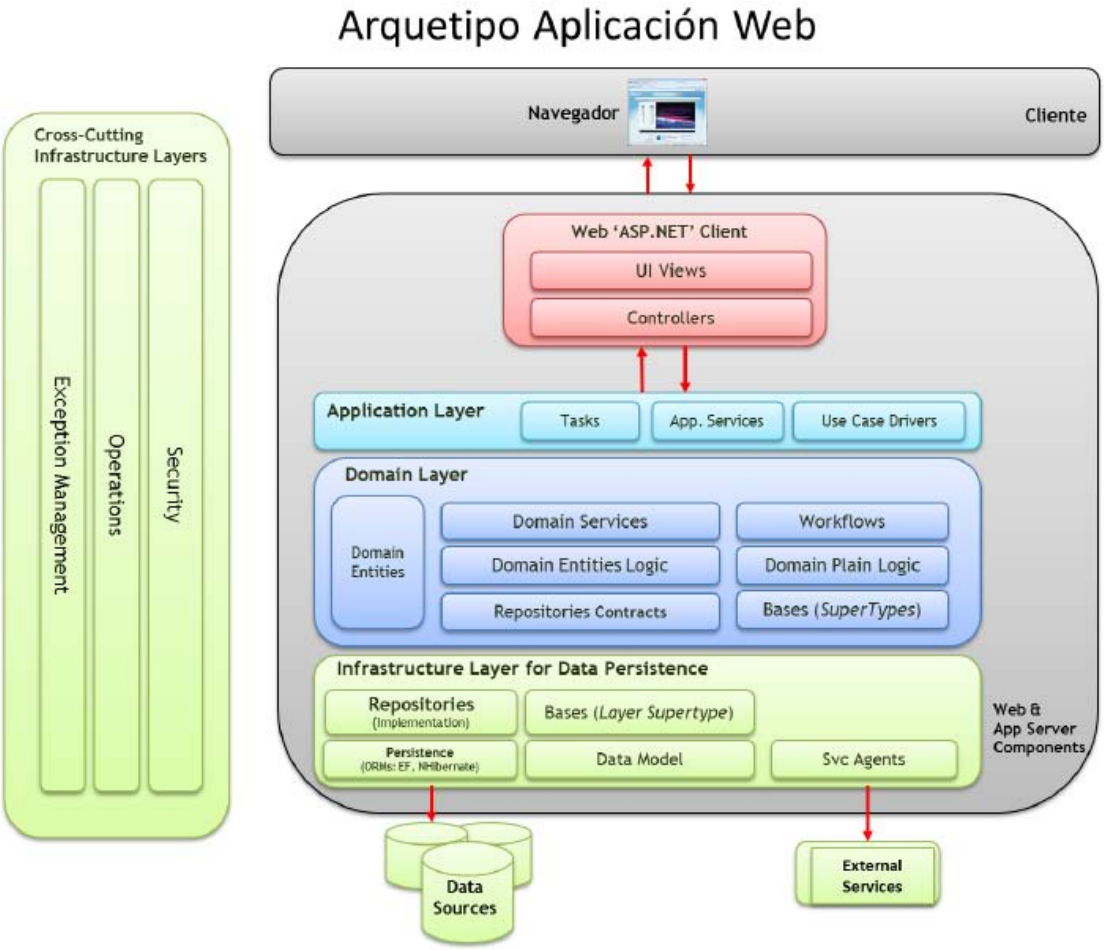


Figura 3-3. Arquetipo Aplicación Web (de la Torre, Zorrilla, Calvarro, & Ramos, 2010).

El arquetipo puede incluir varios estilos arquitectónicos para cubrir los aspectos relevantes de ese tipo de aplicación.

Cuando se pretende construir o reestructurar un sistema del cual se conoce el tipo de aplicación entonces se puede seleccionar un arquetipo adecuado a dicho tipo como punto de partida para el diseño de la arquitectura de software del sistema en cuestión.

CAPÍTULO 4 DISEÑO DE LA ARQUITECTURA

El sistema CAPRA es una implementación en software del conocimiento experto sobre la evaluación de riesgos naturales. Esta aportación científica se destaca por ser una plataforma de software de código abierto con la intención de ser software modular y extensible.

Sin embargo, la arquitectura de dicho sistema carece de un diseño en capas, presenta áreas de mejora en su diseño orientado al dominio y no cuenta con documentación pública de su estructura y comportamiento.

Estas problemáticas requieren ser resueltas mediante una nueva versión del sistema CAPRA, en específico con una reestructuración a su arquitectura de software, para con ello sentar la base para futuras versiones que le permitan evolucionar hacia tecnologías como los dispositivos móviles y aplicaciones en la nube.

Como parte del entendimiento del sistema experto, en los tres capítulos previos se plantearon los fundamentos teóricos de las disciplinas: evaluación de riesgos naturales, sistemas expertos y arquitectura de software; con esto se abre paso al tema central de este trabajo, que consiste en presentar una nueva arquitectura de software para el sistema experto CAPRA.

La solución planteada en esta tesis se ha dividido en dos partes:

- El diseño de una nueva arquitectura de software para CAPRA, que se detalla en este capítulo 4.
- Y la implementación de dicho diseño para generar una nueva versión ejecutable del sistema experto CAPRA, descrita en el capítulo 5.

Para obtener el diseño de la nueva arquitectura se adoptó el proceso de diseño de arquitectura publicado por De la Torre (2010).

Con esa directriz, este capítulo inicia con la descripción de dicho proceso, seguido de la ejecución de cada uno de sus pasos y finaliza con la presentación de la arquitectura de CAPRA, documentada mediante modelo de arquitectura de software “4+1” de Kruchten (1995) basado en diagramas de UML (Unified Modeling Language).

PROCESO DE DISEÑO DE LA ARQUITECTURA DE SOFTWARE

De la Torre presenta un proceso iterativo e incremental de 5 pasos para generar la arquitectura de un sistema (ver figura 4-1). Esas características permiten que el proceso pueda ser adoptado para el desarrollo de sistemas con metodologías tanto ágiles como en cascada.

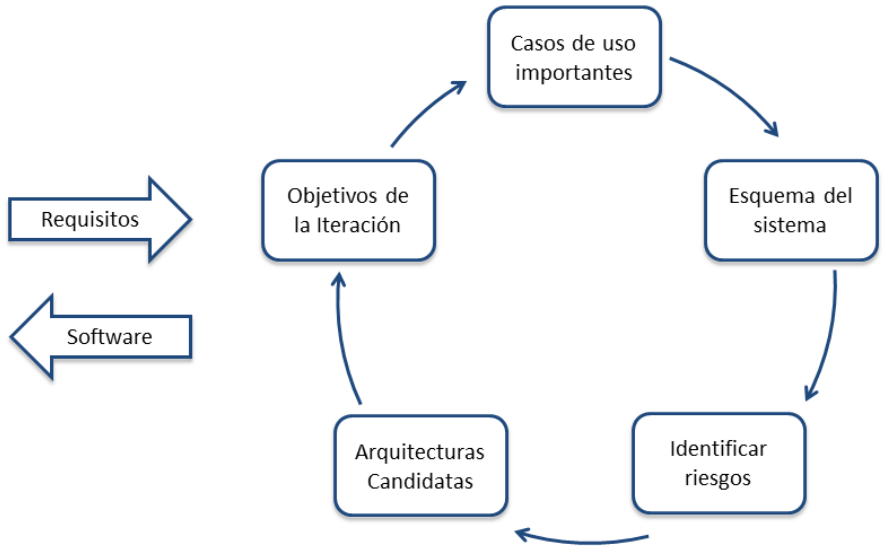


Figura 4-1 Proceso de diseño de la arquitectura (de la Torre, Zorrilla, Calvarro, & Ramos, 2010).

Este proceso tiene como partida los siguientes insumos:

- Casos de uso o historias de usuario
- Requisitos funcionales y requerimientos no funcionales
- Restricciones tecnológicas y de diseño
- Entorno de despliegue propuesto

A continuación se describe brevemente cada uno de los pasos del proceso y en las siguientes secciones se obtendrán los insumos y se ejecutará cada paso para lograr el diseño de la arquitectura del sistema CAPRA.

Identificar los objetivos de la iteración

Este primer paso tiene por objetivo entender el entorno del sistema a diseñar, para ello se analizan las restricciones obtenidas de los requerimientos no funcionales y se decide el alcance de la arquitectura así como el tipo de documentación a generar y a quién va dirigido.

Al finalizar esta fase se debe contar con la lista de objetivos y tareas de la iteración, los recursos que participan, el esfuerzo y tiempo requerido para el resto del proceso.

Seleccionar los casos de uso arquitecturalmente importantes

En este paso se debe obtener la lista de casos de uso que debe cubrir la arquitectura en diseño, en donde un caso de uso representa una parte indivisible de la funcionalidad de negocio que un usuario realiza mediante el uso del sistema.

Esta lista debe contener solo un subconjunto del total de casos de uso; por este motivo se debe ponderar la importancia de cada caso de uso de acuerdo con los siguientes criterios:

- Por su importancia dentro de la lógica de negocio, la cual está dada por la frecuencia de utilización o el valor que le asigna el cliente.
- Por su importancia dentro de la arquitectura, la cual se identifica por la afectación sobre la arquitectura ante el supuesto de no incluir dicho caso de uso.
- Por su importancia al incluir requisitos de calidad como: seguridad, disponibilidad, tolerancia a fallas u aspectos horizontales de la arquitectura.

Se puede diseñar la arquitectura en más de una iteración dependiendo de tamaño del sistema, donde el tamaño es función de la cantidad y complejidad de los casos de uso.

Dicha lista de casos de usos más importantes será el punto de partida para la validación de la arquitectura una vez terminado su diseño.

Realizar un esquema del sistema

En este paso se decide el tipo de aplicación de acuerdo a las restricciones analizadas en el primer paso del proceso. Para ésta decisión es conveniente apoyarse de la tabla 3-1 “Ventajas y consideraciones de los tipos de aplicaciones” y en la figura 3-2 “Cuadrante de decisión para seleccionar un tipo de arquitectura”.

A partir del tipo de aplicación, se debe decidir el tipo de despliegue, estilos arquitectónicos (tabla 3-2) e infraestructura física a utilizar. Al realizar el esquema también se puede optar por seleccionar un arquetipo como punto de partida.

El esquema de la arquitectura debe reflejar estas decisiones tecnológicas, mismas que se utilizarán como guía en la fase de implementación.

Identificar los principales riesgos y definir una solución

El éxito en un proyecto de desarrollo de software se puede ver afectado cuando un requerimiento no funcional no fue considerado en la fase de diseño de la arquitectura y su incorporación tiene un fuerte impacto de esfuerzo o en la fecha de término.

Los requerimientos no funcionales son las propiedades del sistema que deben ser resueltos por la arquitectura por ejemplo: alta disponibilidad, interoperabilidad, flexibilidad, mantenimiento, rendimiento, seguridad, robustez.

Identificar los principales riesgos se traduce en considerar cada requerimiento “no funcional” como un riesgo, el cual debe mitigarse durante el diseño de la arquitectura a través de elementos arquitectónicos transversales.

Los elementos arquitectónicos transversales también son llamados sistemas de soporte que cubren aspectos como autenticación, autorización, caché, configuración, excepciones, bitácora, validación e instrumentación. Dichos elementos transversales deben integrarse como parte de la solución en las arquitecturas candidatas.

Crear arquitecturas candidatas

Finalmente, aquí se concentra el resultado de los pasos previos en una arquitectura candidata.

En caso de existir varias arquitecturas candidatas entonces se selecciona la mejor valorada con base en la funcionalidad de negocio que implementa y la relevancia de los riesgos que mitiga.

La arquitectura candidata se crea utilizando un lenguaje de modelado como ADL (Architecture Description Language), UML, IEEE 1471 o el modelo vista "4+1".

ARQUITECTURA SOFTWARE ACTUAL DEL SISTEMA CAPRA VERSION 2.0

El análisis de la estructura actual del sistema experto CAPRA es el punto de partida para obtener los cuatro insumos requeridos por el proceso de diseño de De la Torre.

La versión actual del sistema CAPRA está publicada como plataforma abierta en el sitio web www.ecapra.org . Está desarrollado en Visual Basic .Net y su código fuente es descargable del sitio web bajo la versión 2.0.

Requerimientos funcionales

Respecto a la funcionalidad de CAPRA, sólo existe información pública sobre dos casos de uso principales documentados como video tutorial, en donde se muestra paso a paso la interacción usuario-sistema. Para el objetivo que ocupa al presente, estos dos casos de uso se toman como los requerimientos funcionales esenciales para la evaluación de riesgos naturales.

Clave	Nombre del caso de uso
CDU-1	Cálculo del riesgo sísmico para un único escenario
CDU-2	Cálculo del riesgo sísmico para un conjunto de riesgo estocástico

Tabla 4-1 Lista de casos de uso del sistema CAPRA v2.0 (Elaboración propia, 2014)

Ambos casos de uso necesitan de tres tipos de archivos estandarizados como insumo para realizar el cálculo de riesgo, siendo estos archivos de: amenaza, vulnerabilidad y datos de exposición.

Estructura de componentes

El sistema CAPRA está desarrollado con el lenguaje de programación Visual Basic .Net y su código fuente está formado por 1 ejecutable y 19 bibliotecas (DLL) cuyas dependencias se muestran en la siguiente figura 4-3. Adicionalmente el código cuenta con 7 referencias a bibliotecas de archivo cuyo código fuente no fue publicado en el sitio web del sistema.

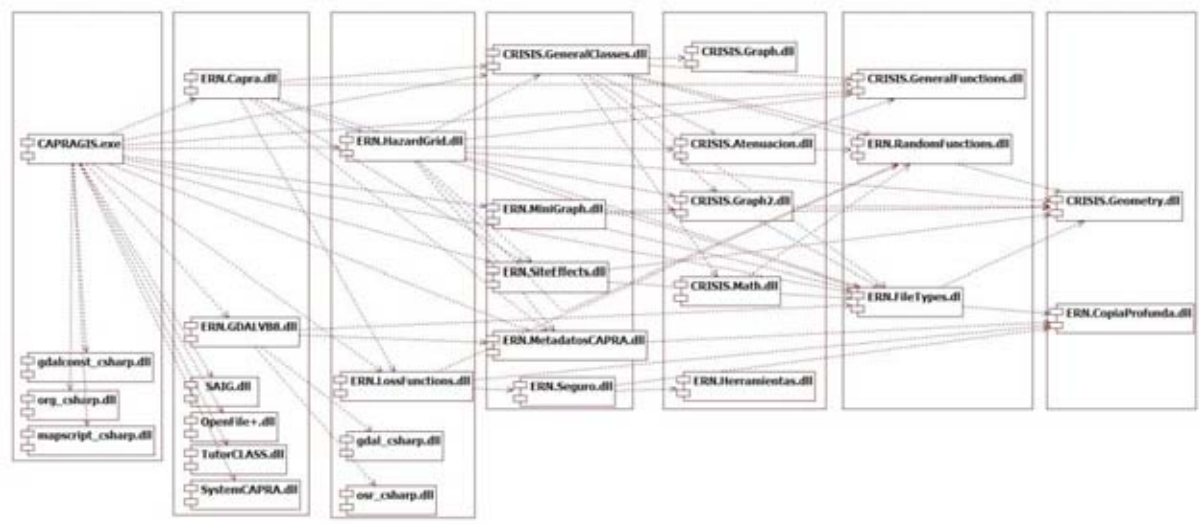


Figura 4-2 Diagrama de componentes del sistema CAPRA v2.0 (Elaboración propia, 2014). Esta figura se presenta a página completa en el anexo 1 de este trabajo para su mejor lectura.

En el diagrama anterior se muestran las dependencias existentes entre los 20 componentes que conforman CAPRA. La cantidad de componentes no tiene relevancia cuando cada componente se forma por los elementos con alta cohesión funcional; sin embargo 7 de los 20 componentes actuales de CAPRA solamente contienen 1 o 2 clases, lo cual implica una baja cohesión a nivel de componente físico (archivo DLL).

Por otra parte, los 7 rectángulos verticales de la figura 4-2 permiten visualizar los niveles de acoplamiento. Se puede apreciar el alto acoplamiento de los componentes contenidos en los 4 rectángulos de la izquierda, es decir, para reutilizar uno de estos componentes, se vuelven indispensables los componentes colocados a su derecha debido a las dependencias con ellos.

Por ejemplo, si en otro proyecto se desea utilizar una clase del componente ERN.MiniGraph cuyo nombre alude a funcionalidad referente a gráficos, entonces por su alto acoplamiento se requiere también de 4 componentes más.

Esta condición actual de baja cohesión y alto acoplamiento conlleva repercusiones negativas en temas como: incorporación de nueva funcionalidad, mantenimiento al código, reutilización de componentes y evolución de su arquitectura.

Organización lógica de las clases

El código del sistema CAPRA corresponde al paradigma orientado a objetos por lo cual su organización lógica se basa en espacios de nombre (namespace).

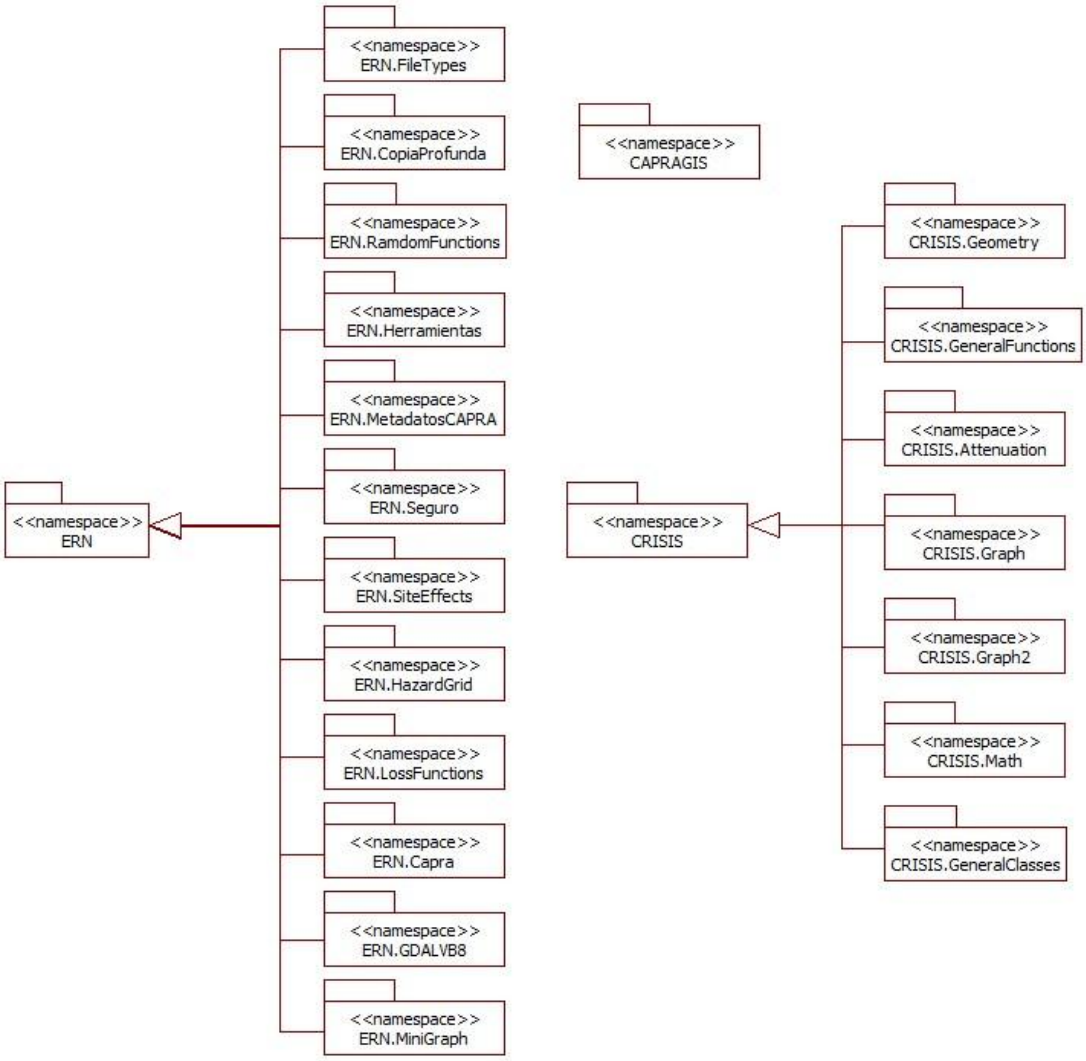


Figura 4-3 Organización lógica de espacios de nombre del sistema CAPRA v2.0 (Elaboración propia, 2014)

Esta organización lógica parece adecuada de primera vista, pero cada espacio de nombres está contenido en un componente. En consecuencia, las dependencias entre los espacios de nombres son las mismas que las dependencias entre componentes mostradas en la figura 4-2. Para reducir tal cantidad de dependencias se requiere una reorganización lógica como parte del diseño de la nueva arquitectura.

Entorno de despliegue

Tras analizar los componentes de CAPRA se determina que el sistema tiene una arquitectura de tipo aplicación de escritorio y su entorno de despliegue es centralizado en el mismo equipo del usuario.

El entorno de despliegue actual es adecuado para un sistema experto con alta demanda de cálculos probabilísticos; por lo tanto se debe mantener igual para aprovechar el poder de cómputo que ofrece el equipo del usuario.

Restricciones tecnológicas

A continuación se listan las restricciones tecnológicas.

Clave	Restricción tecnológica
RT-01	El lenguaje de programación debe ser Visual Basic.Net
RT-02	El software para desarrollo debe ser Visual Studio 2010
RT-03	El layout del archivo de texto que incluye los datos de la amenaza debe permanecer idéntico para respetar el estándar. Archivo con extensión AME.
RT-04	El layout de los dos archivos de texto que incluyen los datos de la vulnerabilidad debe permanecer idéntico para respetar el estándar. Archivo con extensión dat y flv.
RT-05	El layout de los 5 archivos que incluyen los datos de los elementos expuestos debe permanecer idéntico para respetar el estándar. Archivos con extensión dbf, shp, prj, sbx y shx.
RT-06	El control gráfico que encapsula la funcionalidad para manipular mapas geo-referenciados se debe reutilizar para ofrecer la misma usabilidad gráfica de la versión actual.

Tabla 4-2 Lista de restricciones tecnológicas

Con la información expuesta en los incisos previos se conforma los insumos para iniciar los pasos del proceso de diseño de la arquitectura.

GENERAR EL DISEÑO DE LA ARQUITECTURA DEL SISTEMA EXPERTO CAPRA

A continuación se realiza cada uno de los pasos del proceso de diseño mostrado en la figura 4-1 para obtener la arquitectura candidata para el sistema experto CAPRA.

Identificar los objetivos de la iteración

A partir de las restricciones tecnológicas (tabla 4-1) y los requerimientos funcionales de la versión actual se identificaron los siguientes objetivos para el diseño de la arquitectura propuesta:

Clave	Objetivo
OB-01	Diseñar una arquitectura completa que sea implementada y verificada.
OB-02	Soportar la misma funcionalidad de negocio de la versión 2.0 de CAPRA
OB-03	Respetar los layouts de los archivos de amenaza, vulnerabilidad y elementos expuestos.

OB-04	Reestructurar los componentes existentes en una arquitectura en capas y orientada al dominio.
OB-05	Reutilizar la interfaz gráfica de usuario de la versión 2.0
OB-06	Utilizar las tecnologías de Microsoft .Net y Visual Basic.Net como lenguaje de programación.
OB-07	Aprovechar el poder de cómputo del equipo cliente.
OB-08	Aprovechar que los usuarios cuentan con sistema operativo Windows
OB-09	Aprovechar que los usuarios cuentan con permisos para la instalación de CAPRA en sus equipos.

Tabla 4-3 Lista de objetivos para el diseño de la arquitectura

Cabe mencionar que algunos de los objetivos son cubiertos con el diseño de la arquitectura contenido en este capítulo, en tanto el resto de los objetivos son cumplidos tras finalizar la implementación de la arquitectura, misma que se presenta en el capítulo siguiente.

Seleccionar casos de uso arquitecturalmente importantes

Para realizar este paso se tomó la tabla 4-1 y se calificó cada caso de uso bajo los tres criterios de importancia.

- Importancia dentro de la lógica de negocio- LOG
- Importancia dentro de la arquitectura - ARQ
- Importancia como requisito de calidad - CAL

La calificación se basa en la escala de 0 a 3, donde 0 es nada importante, 1 es poco importante, 2 es importante y 3 es muy importante.

En la tabla siguiente se listan los casos de uso importantes para el diseño de la arquitectura.

Clave	Nombre del caso de uso	LOG	ARQ	CAL
CDU-1	Cálculo del riesgo sísmico para un único escenario.	3	3	2
CDU-2	Cálculo del riesgo sísmico para un conjunto de riesgo estocástico.	3	3	3

Tabla 4-4 Lista de casos de uso arquitectónicamente importantes

Realizar el esquema del sistema

El primer paso en la definición del esquema del sistema es la selección del tipo de aplicación.

De acuerdo con el objetivo OB-09 no existe restricción para instalar software en los equipos del cliente y el objetivo OB-08 indica que su sistema operativo es homogéneo, en consecuencia, si utilizamos el cuadrante de decisión para seleccionar un tipo de arquitectura

(figura 3-12), entonces podemos seleccionar cualquiera de los tipos de aplicación del cuadrante 1 así como del resto de los cuadrantes.

Sin embargo, algunos tipos de aplicación no cumplen con restricciones tecnológicas; para dar claridad a ello se genera la siguiente tabla.

Cuadrante	Tipo de aplicación	Restricción u objetivo incumplido
1	Aplicación de escritorio	Ninguno
1	Aplicación móvil instalada en dispositivo	OB-05
2	Aplicación RIA	RT-01, OB-05
3	Aplicación móvil vía web	RT-06, OB-05, OB-07
4	Aplicación web multi-browser	RT-06, OB-05, OB-07
4	Aplicación cloud computing	RT-06, OB-05, OB-07

Tabla 4-5 Tipos de aplicación candidatas (Elaboración propia, 2014)

Derivado de lo anterior, la aplicación de tipo escritorio no se contrapone con los objetivos ni con las restricciones tecnológicas plateadas. Por lo tanto, este tipo de aplicación será la base para el esquema del sistema.

El siguiente diagrama muestra el esquema de la arquitectura propuesta, el cual incluye el tipo de aplicación, topología de despliegue y tipos de archivos requeridos.

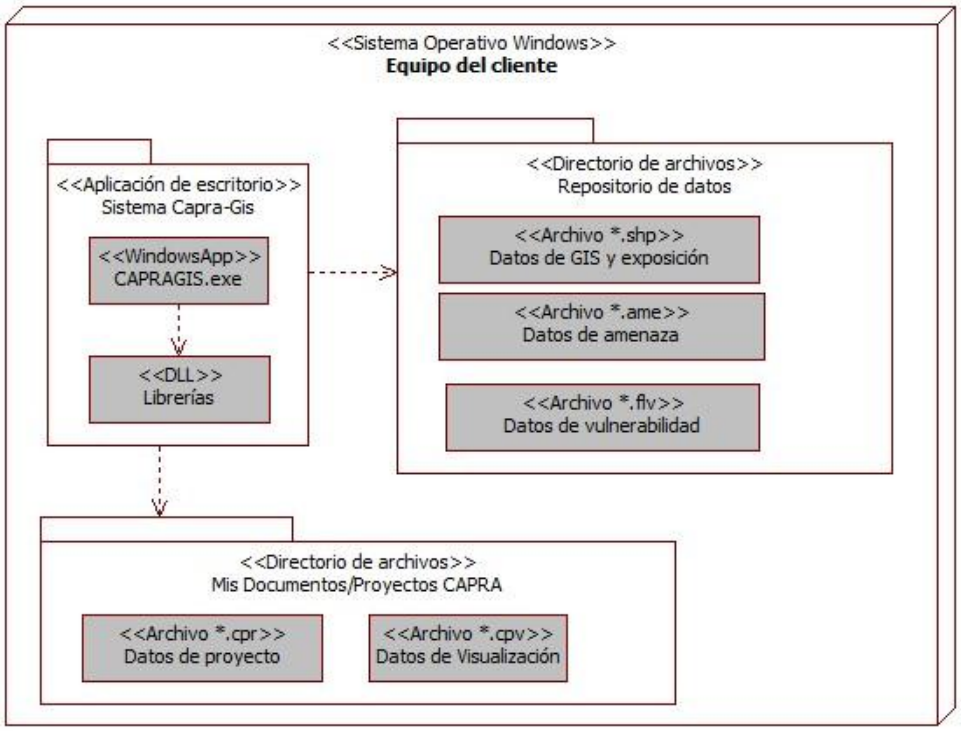


Figura 4-4 Esquema del sistema CAPRA tipo aplicación de escritorio (Elaboración propia, 2014).

Identificación de los principales riesgos

Parte fundamental de este trabajo es generar una nueva arquitectura para el sistema experto existente; entonces los riesgos se centran en aquello que podría impedir el éxito de la implementación de dicha arquitectura. Por esa razón, los riesgos identificados son:

- Afectación funcional. Debido a la generación de defectos en el código durante la reestructuración del mismo.
- Imposibilidad de separar en capas. Debido a al alto acoplamiento en la organización actual del código.
- Separación de capas incompleta. Debido a la dependencia con bibliotecas de las cuales no se tenga el código fuente.
- Implementación incompleta. Esto en el caso de que el código fuente esté incompleto.
- Imposibilidad de validar resultados de nueva versión. Esto en el caso de que los cálculos de riesgo arrojados por el código fuente no coincidan con los del ejecutable contenido en el instalador.

Estos riesgos serán mitigados durante la fase de implementación descrita en el capítulo siguiente; sin embargo, se deben mantener presentes durante esta fase de diseño de la arquitectura.

Crear arquitecturas candidatas

En este paso se evalúan las arquitecturas candidatas; sin embargo las restricciones tecnológicas acotaron a sólo una arquitectura candidata como se presenta en la tabla 4-5 tipos de aplicación candidatas (elaboración propia, 2014).

El tipo de aplicación seleccionado y su respectivo arquetipo serán la base de la arquitectura propuesta y definida en la siguiente sección.

DOCUMENTACIÓN DE LA ARQUITECTURA PROPUESTA

Para representar el diseño de la arquitectura propuesta se ha optado por utilizar UML como lenguaje de modelado y el modelo de vista "4+1" propuesto por Kruchten para modelar la arquitectura del sistema. Este se muestra en la figura 4-5.

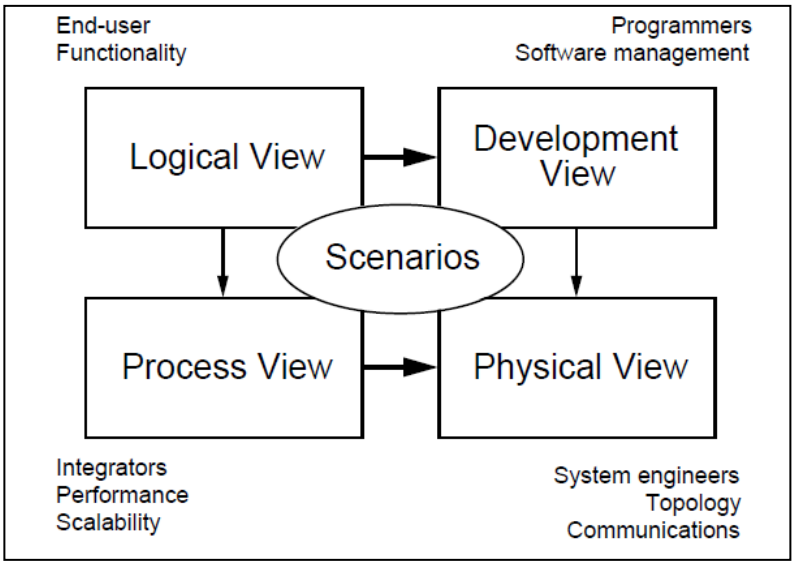


Figura 4-5 El modelo de vista "4+1" (Kruchten, 1995)

El modelo "4+1" sirve para describir la arquitectura de software de un sistema a través de 5 vistas que permiten tratar por separado las preocupaciones de varios stakeholders, entre ellos: usuarios, desarrolladores, ingenieros de infraestructura, administrador del proyecto y analistas.

A continuación se presenta la arquitectura propuesta modelada mediante cada una de las vistas.

Vista de Escenarios

Esta vista representa los casos de uso que conforman el sistema, los cuales encapsulan la funcionalidad del negocio o del sistema experto, en este caso. Con ello se contempla el objetivo OB-02 Soportar la funcionalidad de negocio de la versión 2.0 de CAPRA.

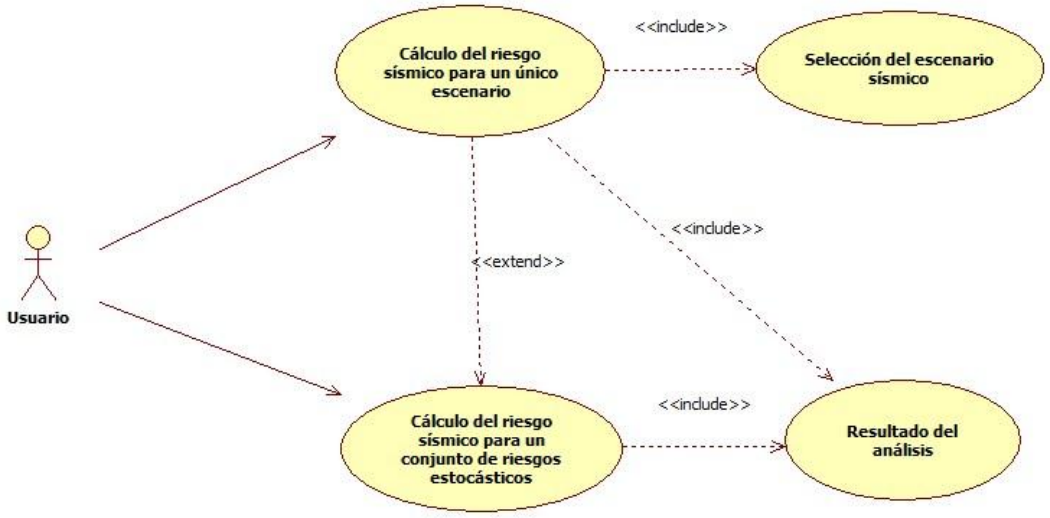


Figura 4-6 Diagrama de casos de uso importantes del sistema CAPRA (Elaboración propia, 2014)

Vista lógica

Esta vista contiene el modelo de dominio, el cual se compone por diagramas de clases que representan a todos aquellos términos y conceptos que los usuarios utilizan y dominan como lenguaje exclusivo de su área de conocimiento, en este caso de la evaluación de riesgos naturales.

El siguiente diagrama se debe considerar un modelo de domino preliminar debido a que el modelo de dominio se tendrá después de obtener todas las entidades de negocio existentes en la versión actual de CAPRA durante la etapa de implementación de la arquitectura.

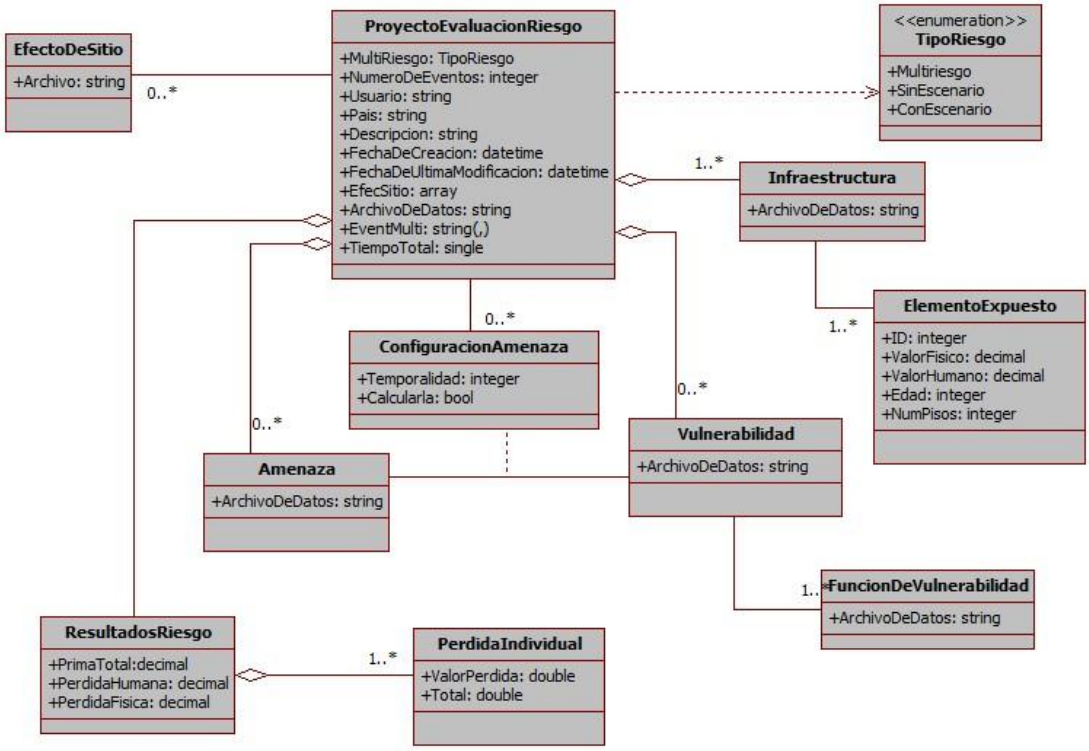


Figura 4-7 Modelo de dominio preliminar. Arquitectura propuesta para el sistema CAPRA (Elaboración propia, 2014)

Con este enfoque se cubre el objetivo OB-04 Reestructurar en una arquitectura orientada al dominio.

Vista de desarrollo

Esta vista describe la estructura del software desde la perspectiva del desarrollador. Esta vista se conforma de un diagrama de paquetes que representa la organización lógica en espacios de nombre y capas, atendiendo al objetivo OB-04 Reestructurar en una arquitectura en capas. Además de un diagrama de componentes por cada una de las tres capas.

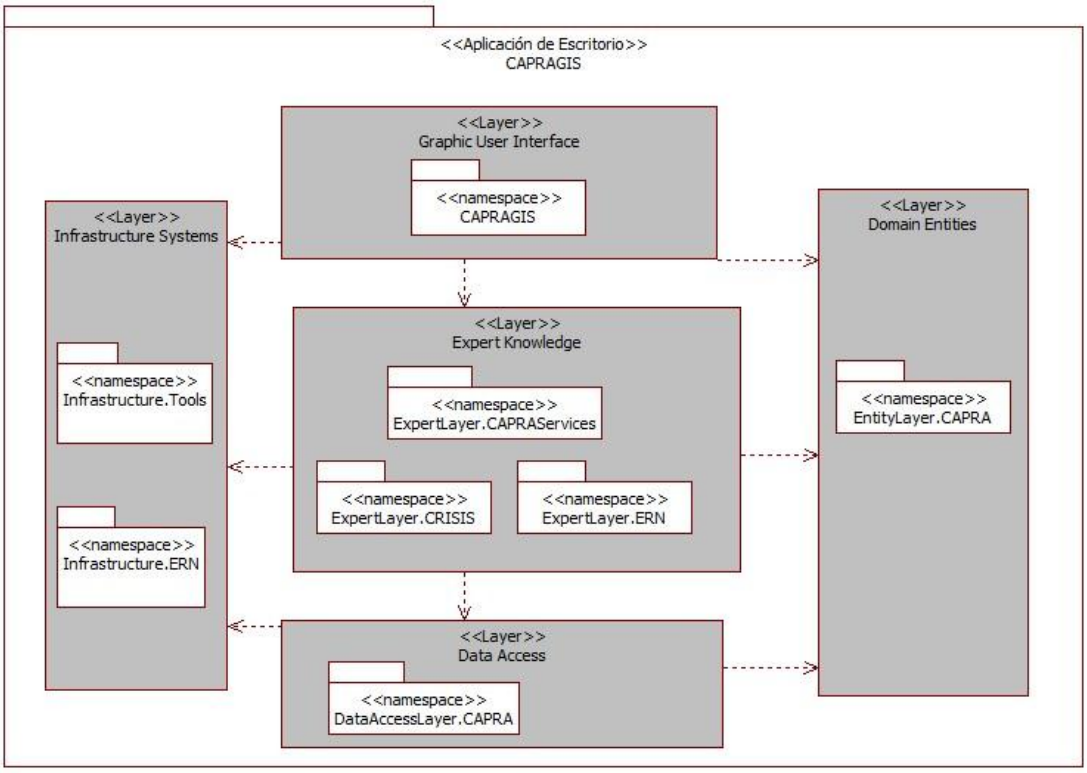


Figura 4-8 Diagrama de capas y espacios de nombre. Arquitectura propuesta para el sistema CAPRA (Elaboración propia, 2014).

Del diagrama anterior cabe mencionar, que a la capa intermedia suele llamársele Business Layer (capa de negocio); en este caso se ha nombrado como Expert Knowledge Layer para focalizar que el conocimiento de expertos y especialistas en evaluación de riesgos naturales se concentra en esta capa.

A continuación se detalla cada una de las capas de la figura 4-8 mediante el uso de diagramas de componentes, donde cada componente representa a un archivo ejecutable o biblioteca DLL.

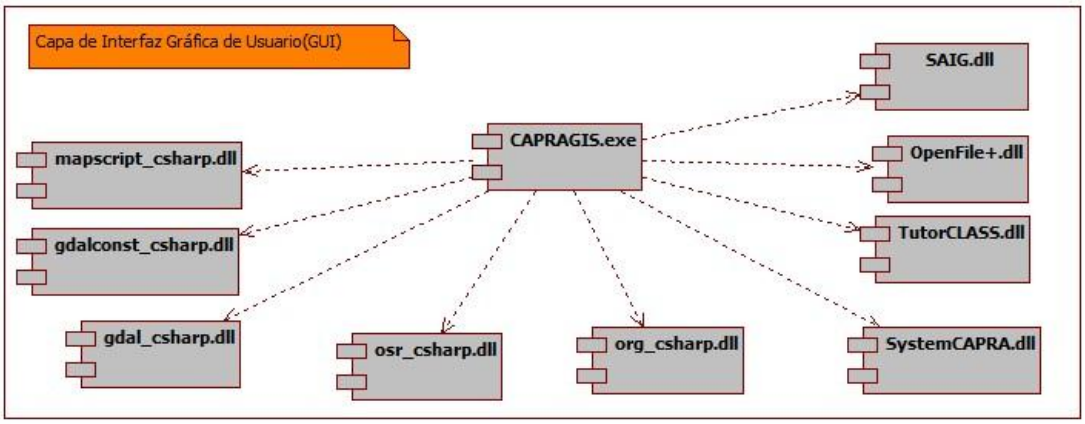


Figura 4-9 Diagrama de componentes de la capa GUI. (Elaboración propia, 2014).

En la figura 4-9 se muestra el componente CAPRAGIS.exe el cual contiene las clases que son responsables de obtener y presentar información de las pantallas que conforman la interfaz gráfica de usuario. El resto de los componentes son bibliotecas compiladas de las cuales no se obtuvo el código fuente por lo tanto fueron consideradas parte de esta capa por la alta dependencia de CAPRAGIS.exe hacia ellas.

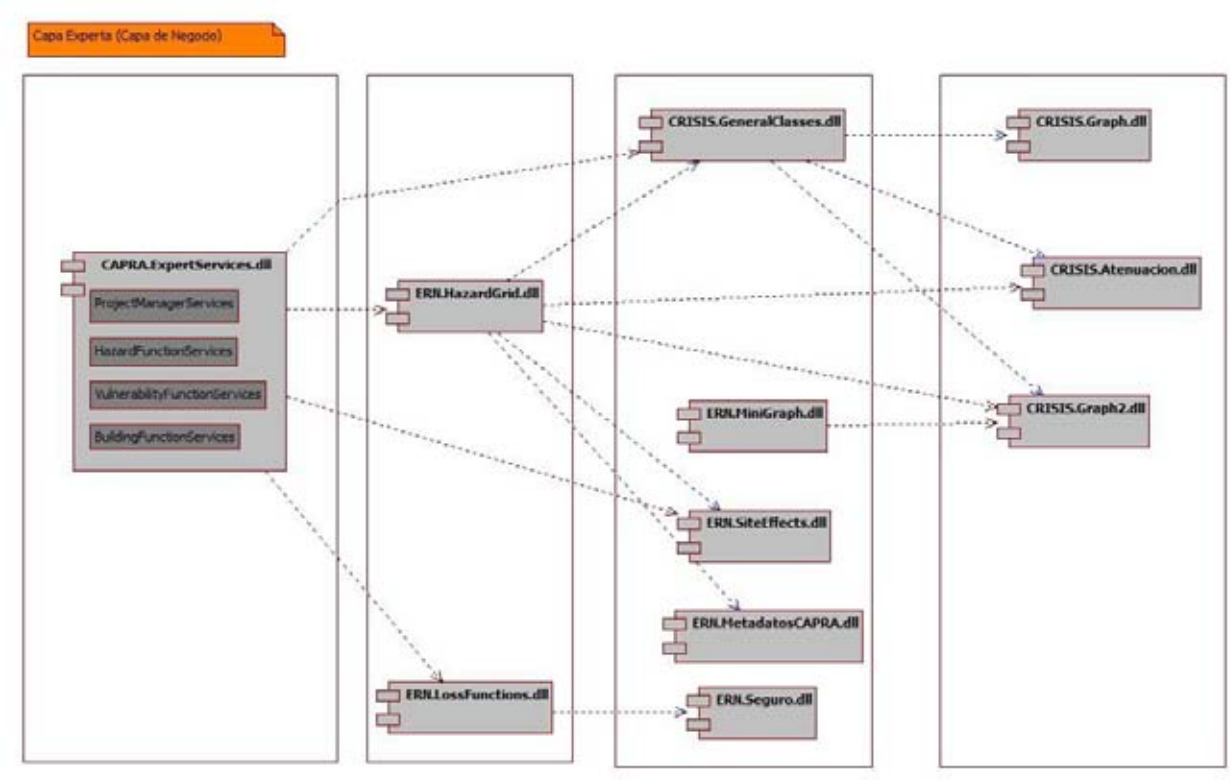


Figura 4-10 Diagrama de componentes capa de conocimiento experto o capa de negocio. (Elaboración propia, 2014).

La figura 4-10 presenta los componentes que forman la capa de conocimiento experto, también conocida como capa de negocio. En estos componentes se encuentran las clases que contienen el código fuente de las operaciones, cálculos, dediciones, aplicación de reglas, algoritmos, ecuaciones y demás representaciones del conocimiento especializado sobre temas de amenaza, vulnerabilidad, riesgo, pérdidas que en su conjunto definen el campo de conocimiento de la evaluación de riesgos naturales, tal como se describe en el capítulo 1.

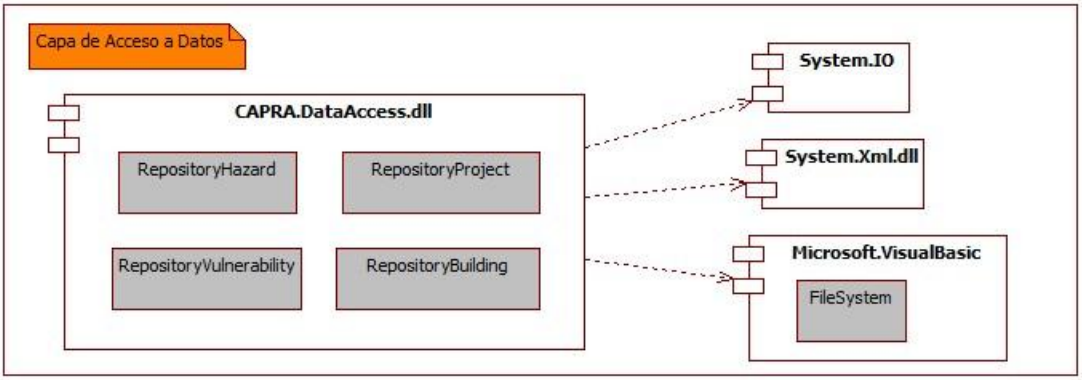


Figura 4-11 Diagrama de componentes de la capa de acceso a datos. (Elaboración propia, 2014).

La figura 4-11 corresponde a la capa de datos, donde resalta el componente CAPRA.DataAccess.dll, el cual está conformado por cuatro clases específicas para controlar el acceso a los repositorios de datos para: amenaza, vulnerabilidad, construcciones y el proyecto. Cada clase tiene por objeto concentrar el código para leer y escribir en las fuentes de datos; que para el caso de CAPRA son archivos XML, DBF, binarios y de texto, lo cuales son manipulados con las componentes de la parte derecha del diagrama.

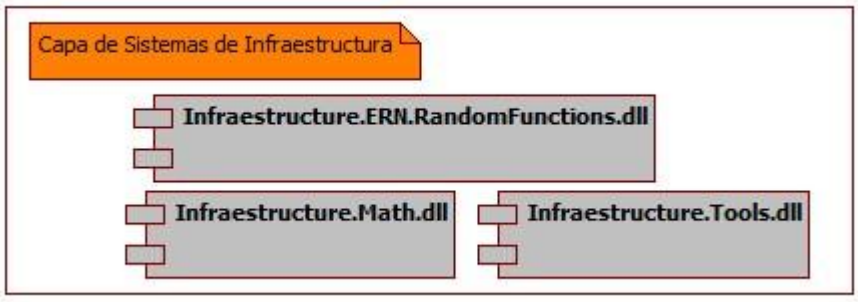


Figura 4-12 Diagrama de componentes de la capa de sistemas de infraestructura. (Elaboración propia, 2014).

La figura 4-11 muestra el detalle de la capa de sistemas de infraestructura, los componentes que la forman tienen por objeto contener la funcionalidad común a varias capas, además no

es exclusiva del conocimiento experto en la evaluación de riesgos. Por lo tanto estos componentes son candidatos para ser reutilizados en otros sistemas expertos.

Vista del proceso

Esta vista captura el comportamiento del sistema en tiempo de ejecución, el cual está conformado por cientos de invocaciones o mensajes entre sus clases.

El diseño detallado de dichas invocaciones se realiza durante la construcción del sistema, sin embargo durante el diseño de la arquitectura es conveniente modelar patrones de invocaciones utilizando diagramas de secuencia donde se muestre la interacción para los procesos básicos como: captura y almacenamiento de información, lectura y presentación de información, y cálculo o procesamiento de información.

En los diagramas de secuencia, la clase EntidadDeNegocio pertenece al modelo de dominio y tiene por objetivo agrupar propiedades con alta cohesión funcional. Esta entidad se utiliza para transferir la información entre las capas por medio de parámetros de entrada a los métodos o respuestas de los mismos.

Cabe aclarar, que en los diagramas subsecuentes cada clase representa de manera general a una o a varias clases dentro de la capa respectiva, de igual forma cada método representa el comportamiento que podría estar encapsulado en 1 o varios métodos de una o varias clases dentro de la misma capa. Lo relevante en estos diagramas son las interacciones esperadas entre las distintas capas.

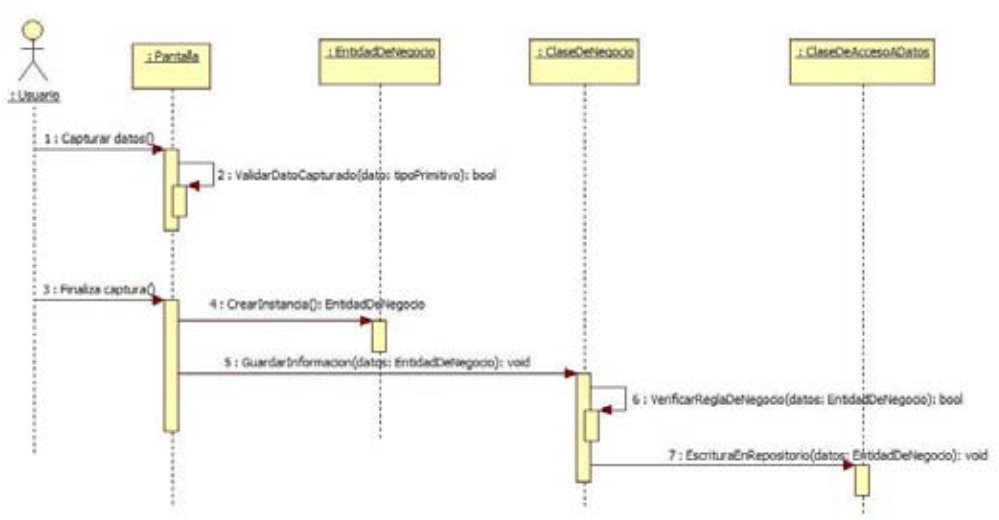


Figura 4-13 Diagrama de secuencia – Proceso de captura y almacenado de información (Elaboración propia, 2014)

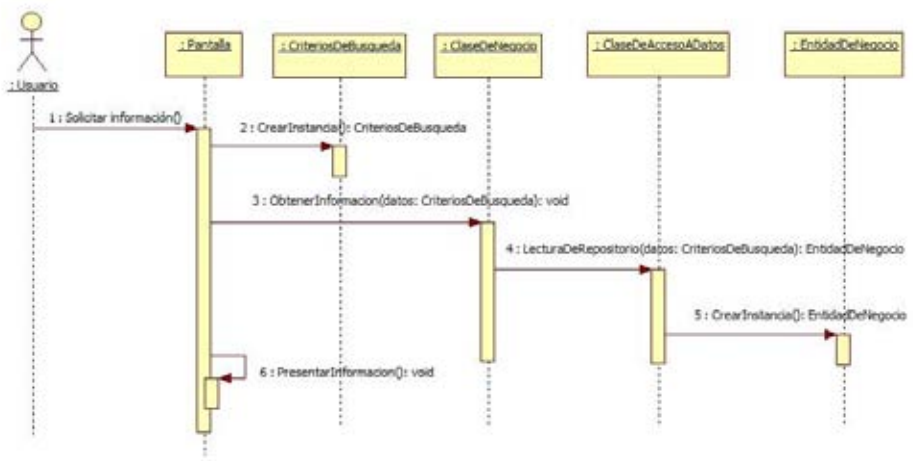


Figura 4-14 Diagrama de secuencia – Proceso de lectura y presentación de información (Elaboración propia, 2014)

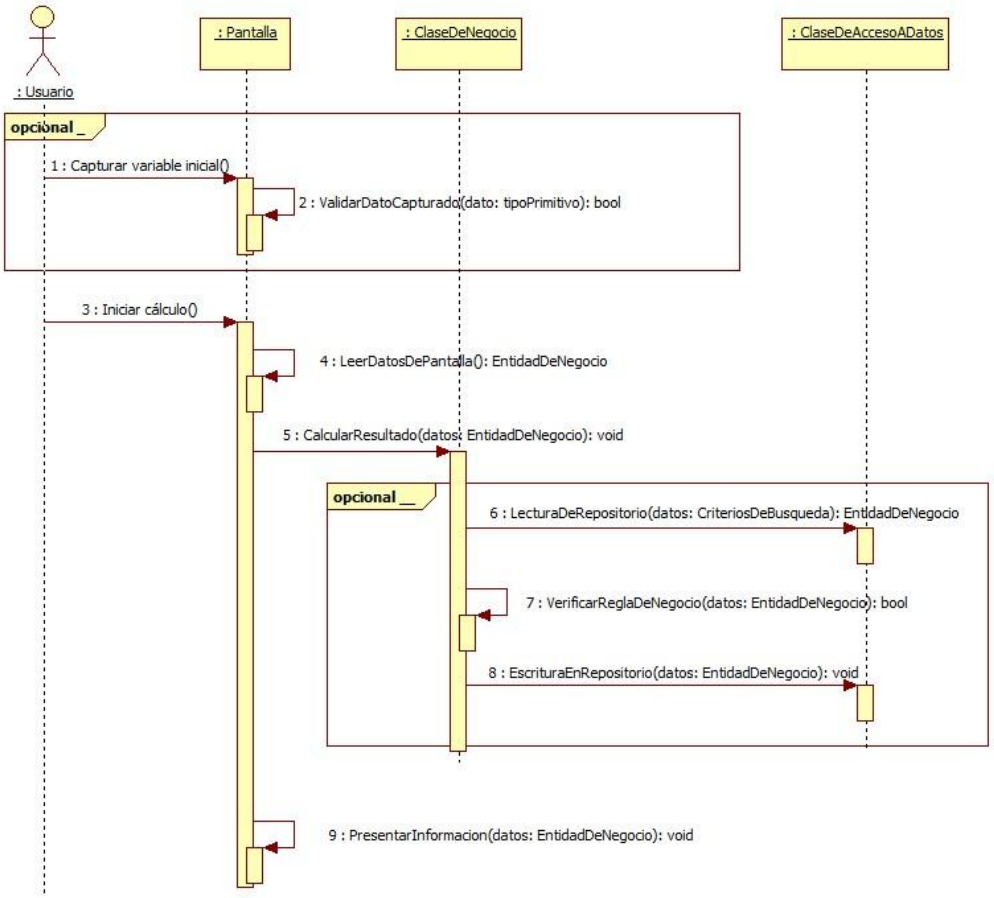


Figura 4-15 Diagrama de secuencia – Proceso de cálculo o procesamiento información (Elaboración propia, 2014)

Vista física

Esta vista describe el mapeo del software a instalar sobre el hardware requerido para ello. El siguiente diagrama de despliegue refleja una instalación centralizada en el propio equipo del usuario para cumplir con los objetivos OB-07 Aprovechar el poder de cómputo del equipo del cliente y OB-09 Aprovechar que los usuarios cuentan con permisos para la instalación de CAPRA en sus equipos.

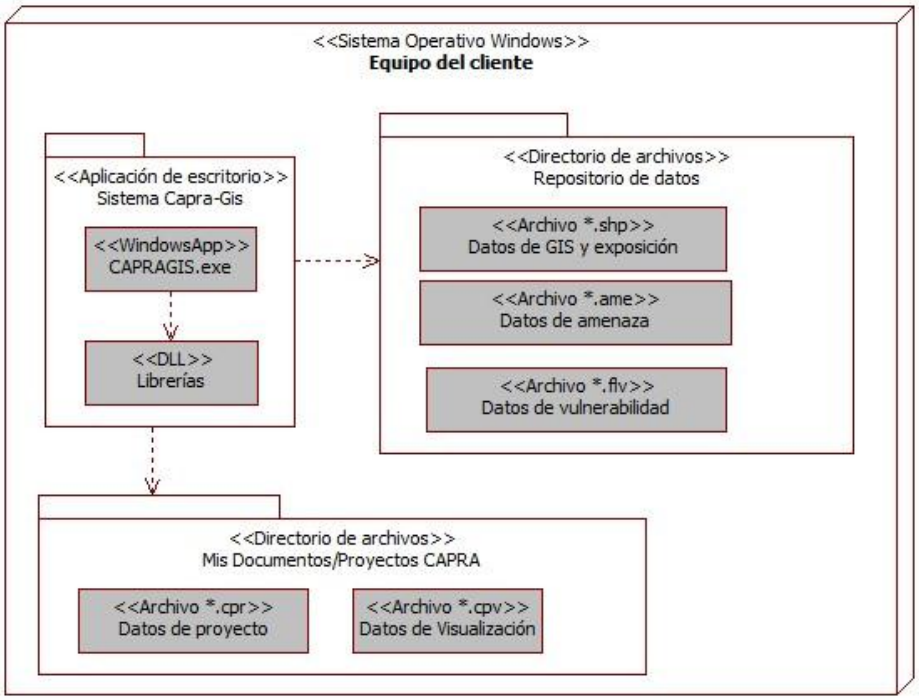


Figura 4-16 Diagrama de despliegue. Arquitectura propuesta para el sistema CAPRA (Elaboración propia, 2014)

Este conjunto de diagramas de UML organizados en 5 vistas definen la arquitectura de software propuesta para el sistema experto CAPRA.

En primer lugar, la vista de escenarios da una visión de conjunto de la funcionalidad que ofrece el sistema al experto en evaluación de riesgos naturales.

La vista lógica contiene un diagrama de clases llamado: modelo de dominio el cual representa los datos y conceptos exclusivos de la evaluación de riesgos que son procesados dentro del sistema.

Por su lado, la vista física ofrece una imagen clara de cuántos elementos de hardware intervienen en el despliegue del sistema.

La vista de desarrollo contiene diagramas de componentes y paquetes que ofrecen los planos principales de la estructura del sistema a construir.

Finalmente, la vista de procesos contiene tres diagramas de secuencia con el comportamiento base al que se apegan la mayoría de las interacciones entre el usuario y el sistema.

Con estas vistas queda conformado el modelo “4+1” atendiendo al objetivo OB-01 de Diseñar una arquitectura completa; misma que será implementada y verificada en el capítulo siguiente.

CAPÍTULO 5 IMPLEMENTACIÓN DE LA ARQUITECTURA

Este capítulo presenta la implementación de la arquitectura diseñada en el capítulo 4; previo a ello se describe la preparación del ambiente de desarrollo y los criterios que validan la correcta implementación de la arquitectura.

La implementación se presenta mediante los resultados obtenidos tras la ejecución de las siguientes actividades: creación de nuevos proyectos, reestructuración de código, renombrado de espacios de nombre y reducción de dependencias. Estas actividades son parte del proceso de reestructuración del código en capas y entidades de negocio, propuesto en este trabajo como un mecanismo para la implementación de la nueva arquitectura.

Al final del capítulo se encuentra la comprobación de la funcionalidad original y se concluye con la validación del cumplimiento de los objetivos planteados durante la fase de diseño de la arquitectura.

PREPARACIÓN PARA LA IMPLEMENTACIÓN

Previo a cualquier modificación, como buena práctica de la ingeniería de software, se debe proceder a obtener una línea base de la funcionalidad y del código que conforman la versión a modificar. La obtención de la línea base en este caso obedece a que la implementación de la nueva arquitectura se realiza sobre una versión existente del sistema CAPRA.

Inventario de código de CAPRA versión 2.0

El sistema CAPRA fue desarrollado con el lenguaje Visual Basic .Net. Los archivos con código fuente tienen extensión “vb” y los asociados a proyectos tienen la extensión “vbproj”; por tanto el inventario se realizó sobre estos dos tipos de archivo más relevantes.

Mediante el uso de software Microsoft LOC Counter se realizó el conteo de archivos y el número de líneas de código fuente (LOC) contenido en ellos, considerando sólo los archivos con extensión “vb”. El resumen del conteo se presenta en la siguiente tabla:

Componente	Proyecto VB.Net	Archivos vb	LOC
CAPRAGIS.exe	CAPRAGIS.vbproj	27	15784
ERN.CAPRA.dll	ERN.CAPRA.vbproj	6	4322
ERN.GDAL.VB6.dll	ERN.GDAL.vbproj	1	49
ERN.HazardGrid.dll	ERN.HazardGRID.vbproj	6	3206
CRISIS.GeneralClasses.dll	Crisis.GeneralClasses.vbproj	3	3734
ERN.MiniGraph.dll	ERN.MiniGraph.vbproj	3	823
ERN.SiteEffects.dll	ERN.SiteEffects.vbproj	3	1071
ERN.MetadatosCAPRA.dll	ERN.MetadatosCAPRA.vbproj	2	498
ERN.Seguros.dll	ERN.Seguro.vbproj	1	617
CRISIS.Graph.dll	Crisis.Graph.vbproj	1	327
CRISIS.Atenuacion.dll	Crisis.Attenuation.vbproj	1	1071
CRISIS.Graph2.dll	Crisis.Graph2.vbproj	1	885
CRISIS.Math.dll	Crisis.Math.vbproj	1	281
ERN.Herramientas.dll	ERN.Herramientas.vbproj	2	287
CRISIS.GeneralFuntions.dll	Crisis.GeneralFunctions.vbproj	3	393
ERN.RandomFuntion.dll	ERN.RandomFunctions.vbproj	8	1550
ERN.FileType.dll	ERN.FileTypes.vbproj	11	4589
CRISIS.Geometry.dll	Crisis.Geometry.vbproj	2	2007
ERN.CopiaProfunda.dll	ERN.CopiaProfunda.vbproj	1	12
ERN.LossFunctions.dll	ERN.LossFunctions.vbproj	3	1543
	TOTAL→	86	43049

Tabla 5-1 Resumen del conteo de LOC de sistema CAPRA versión 2.0

Para el conteo de LOC se consideró como LOC válida toda aquella línea de texto excepto líneas en blanco y líneas que inician con apóstrofe para comentarios de Visual Basic.

Criterios de aceptación funcional

Los criterios de aceptación funcional son el conjunto de condiciones que debe cumplir el sistema para ser aceptado por el usuario como un producto funcional.

Un objetivo de la arquitectura a implementar es “Soportar la misma funcionalidad de negocio de la versión 2.0 de CAPRA” (OB-02). En el alcance de este trabajo considera que la funcionalidad de negocio de dicho objetivo está compuesta por los dos casos de uso arquitectónicamente relevantes de la tabla 4-4.

Por ende los criterios de aceptación funcional de esta implementación consisten en obtener un resultado idéntico al ejecutar ambos casos de uso en la nueva versión 3.0 del sistema CAPRA. En consecuencia, primeramente se requiere documentar el resultado de la ejecución en la versión 2.0, es decir, obtener la línea base funcional.

Línea base funcional

La línea base funcional se obtuvo al ejecutar los dos casos de uso utilizando los archivos de datos del ejemplo "Isla CAPRA" que se encuentran descargables en el sitio de www.ecapra.org (CAPRA, 2012).

Los resultados de los dos casos de uso: CDU-1 y CDU-2, se presentan a la siguiente sección.

Resultados del Escenario 1: CDU-1

A continuación se presentan las imágenes de CAPRA 2.0 tras la ejecución del CDU-1 correspondiente a los resultados del cálculo del riesgo sísmico para un único escenario.

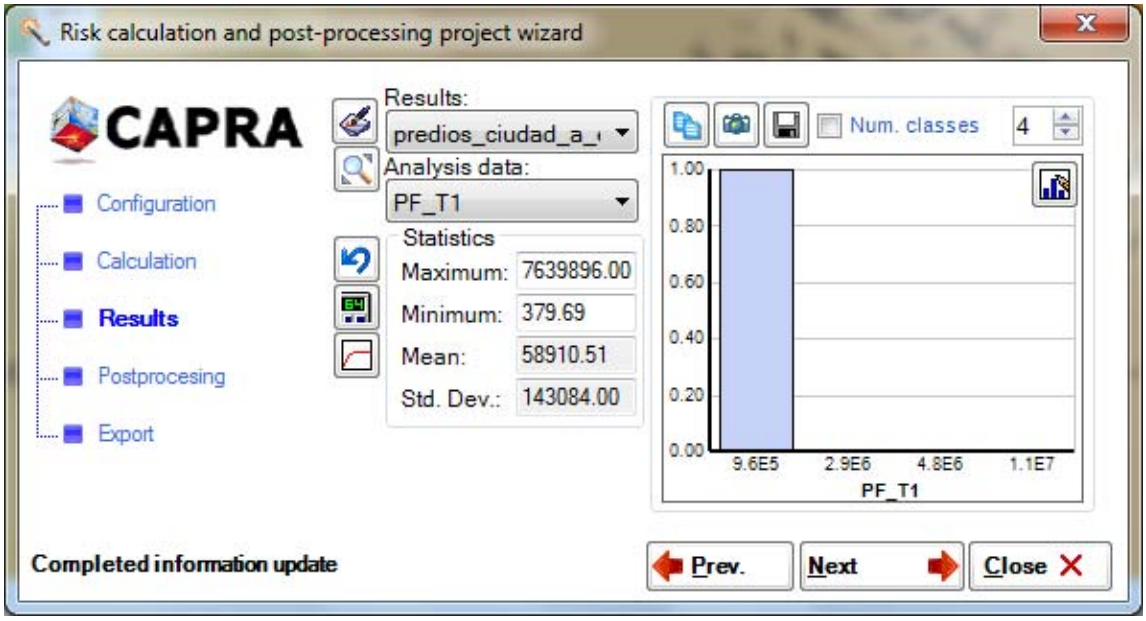


Figura 5-1 Resultado de CAPRA 2.0 para el CDU-1 Escenario 407. Dato analizado: PF_T1.

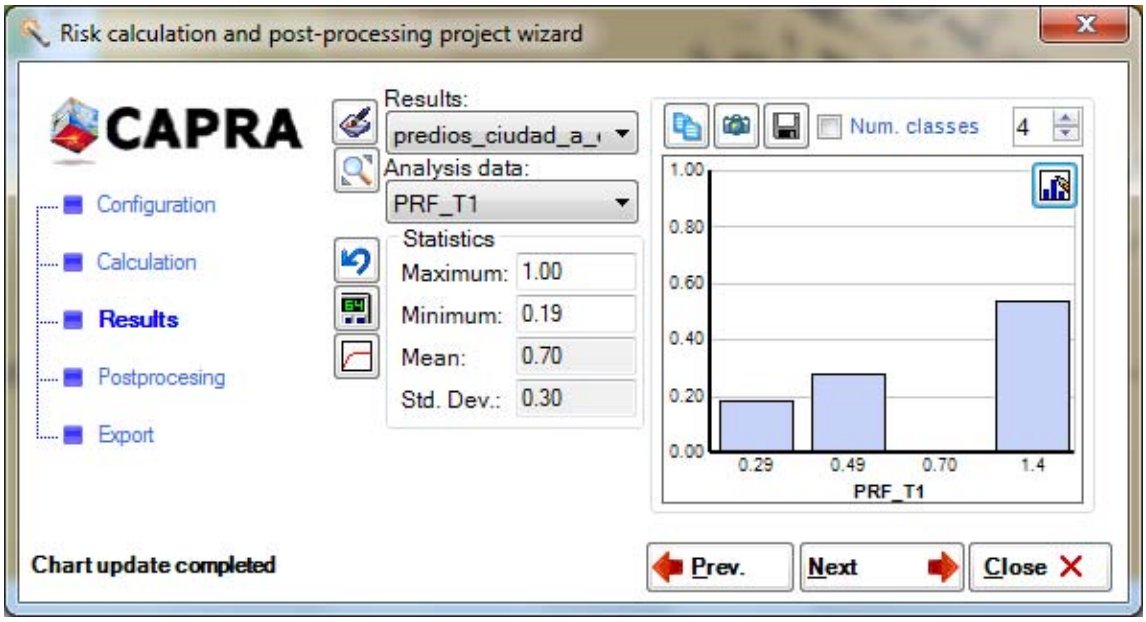


Figura 5-2 Resultado de CAPRA 2.0 para el CDU-1 Escenario 407. Dato analizado: PRF_T1.

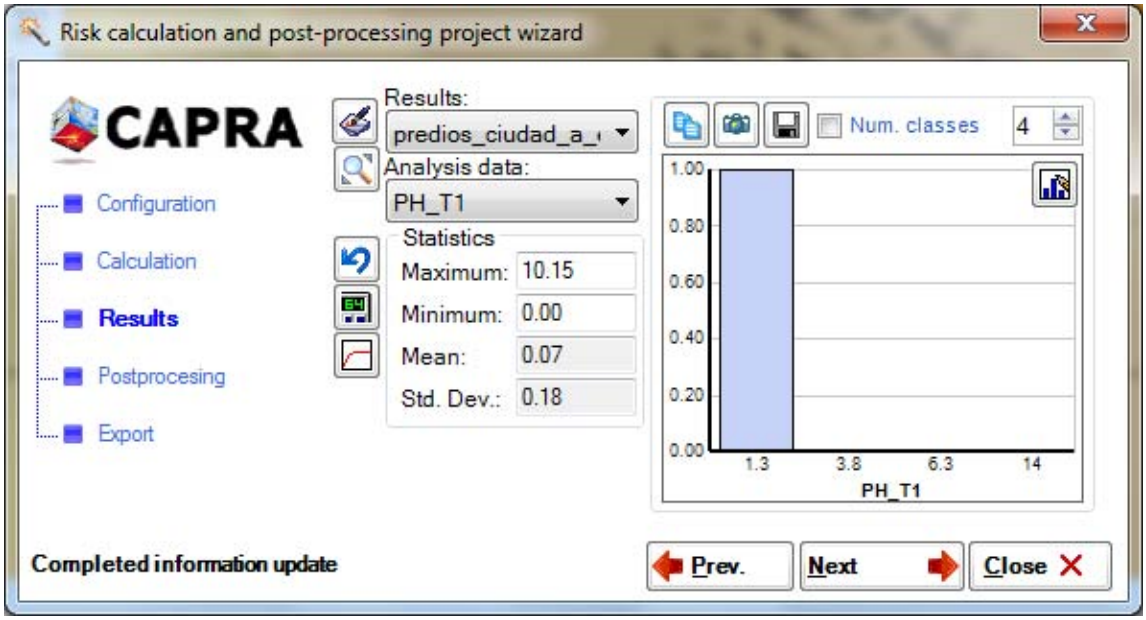


Figura 5-3 Resultado de CAPRA 2.0 para el CDU-1 Escenario 407. Dato analizado: PH_T1.

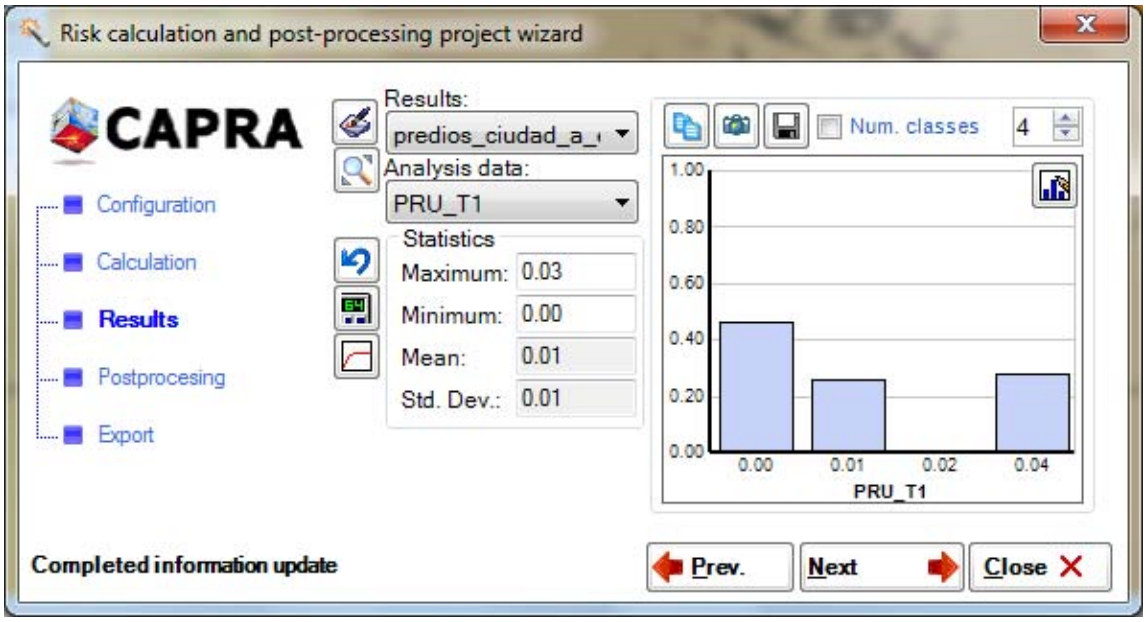


Figura 5-4 Resultado de CAPRA 2.0 para el CDU-1 Escenario 407. Dato analizado: PRU_T1.

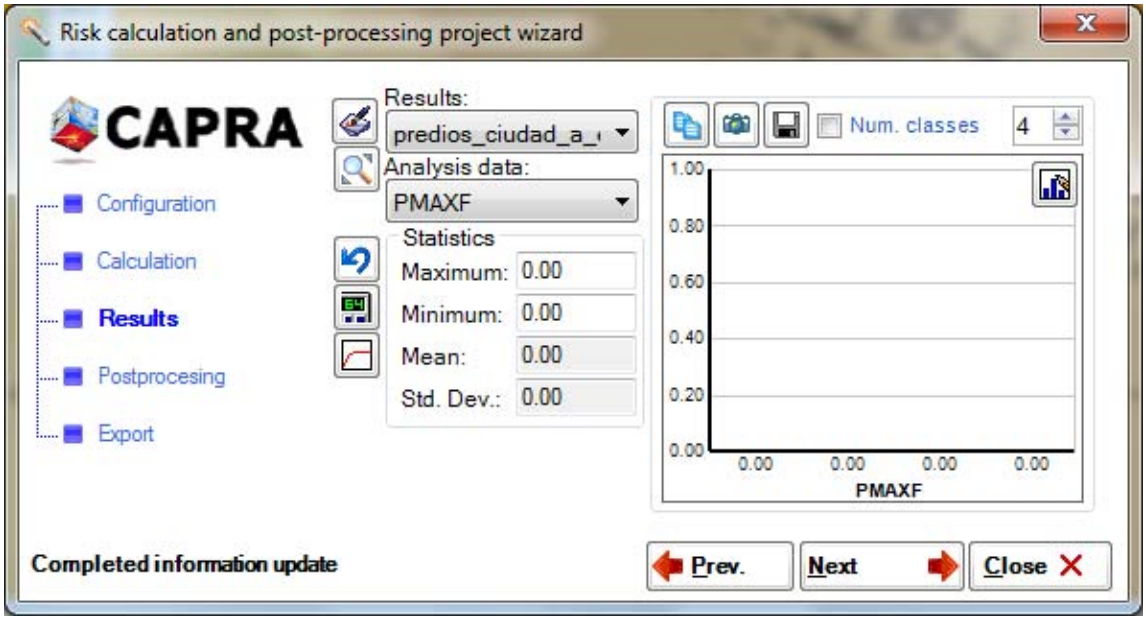


Figura 5-5 Resultado de CAPRA 2.0 para el CDU-1 Escenario 407. Dato analizado: PMAXF.

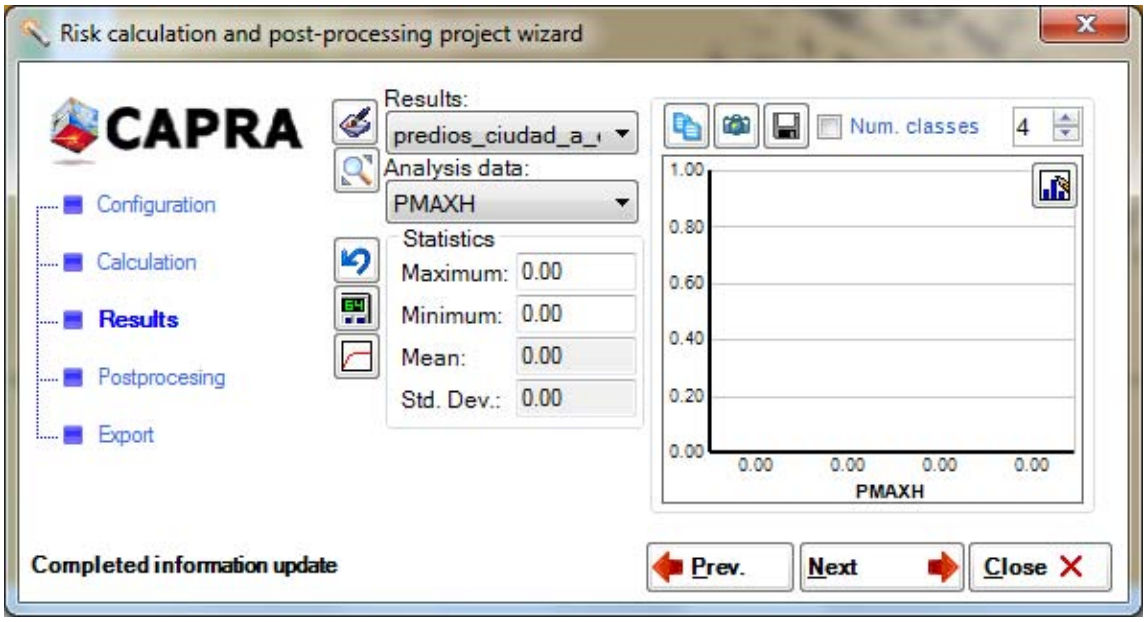


Figura 5-6 Resultado de CAPRA 2.0 para el CDU-1 Escenario 407. Dato analizado: PMAXH.

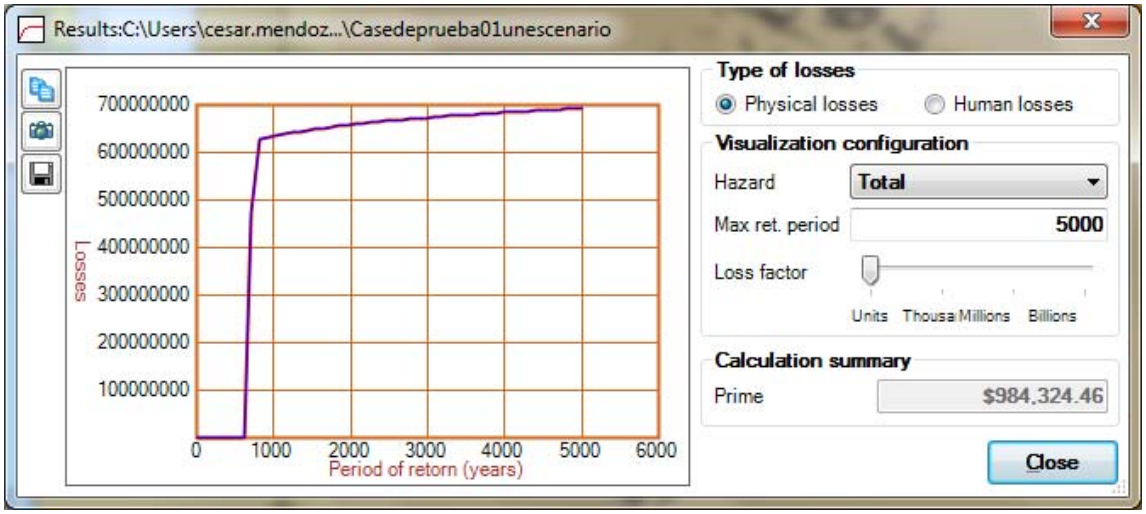


Figura 5-7 Resultado de CAPRA 2.0 para el CDU-1 Escenario 407. Dato analizado: Pérdidas físicas.

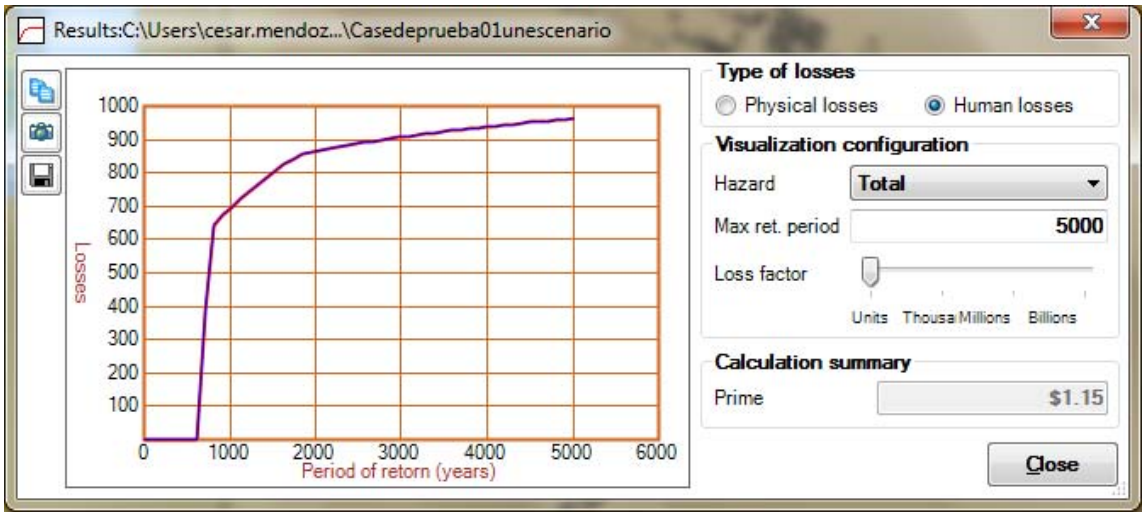


Figura 5-8 Resultado de CAPRA 2.0 para el CDU-1 Escenario 407. Dato analizado: Pérdidas humanas.

Las ocho imágenes anteriores conforman el resultado de la ejecución del caso de uso CDU-1.

Resultados del Escenario 2: CDU-2

A continuación se presentan las imágenes de CAPRA 2.0 tras la ejecución del CDU-2 correspondiente a los resultados del cálculo del riesgo sísmico para un conjunto de riesgo estocástico.

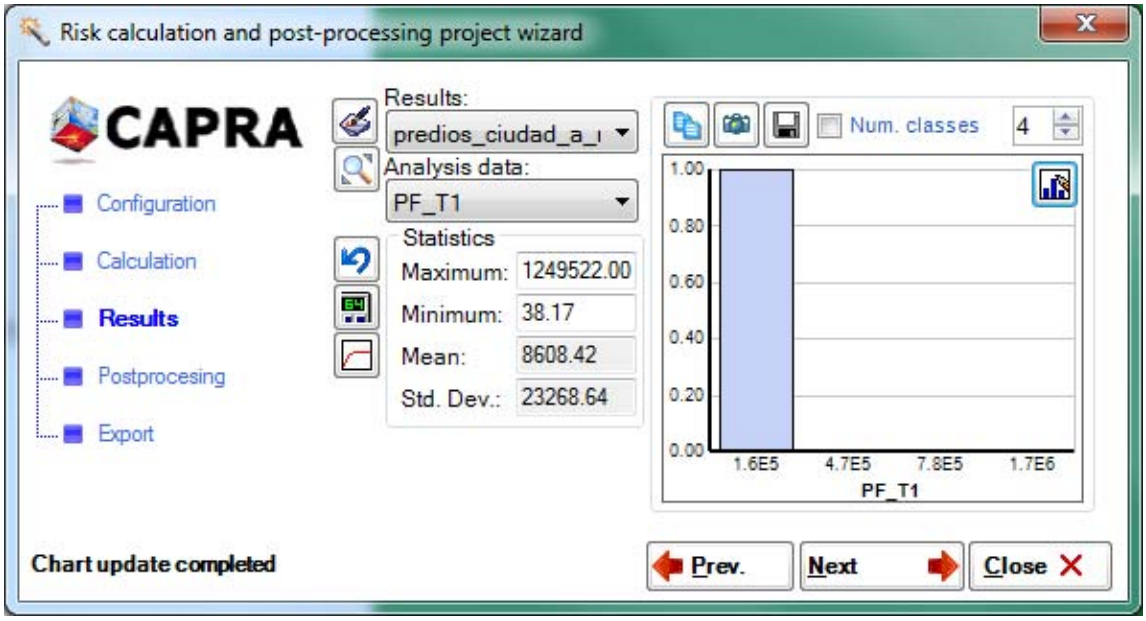


Figura 5-9 Resultado de CAPRA 2.0 para el CDU-2 Todos los escenarios. Dato analizado: PF-T1.

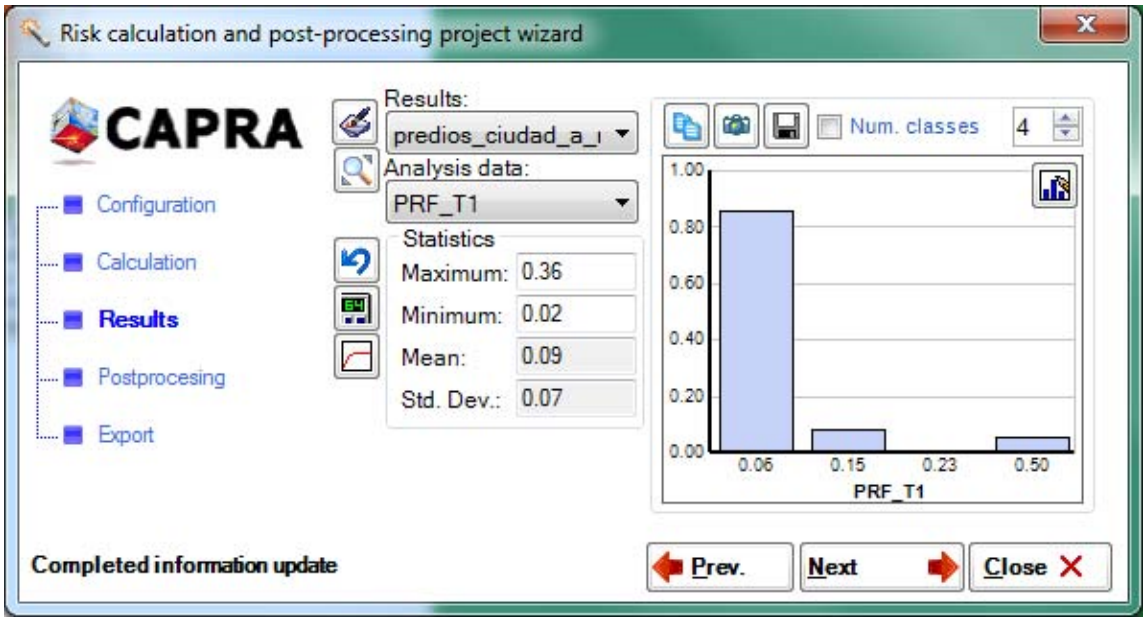


Figura 5-10 Resultado de CAPRA 2.0 para el CDU-2 Todos los escenarios. Dato analizado: PRF-T1.

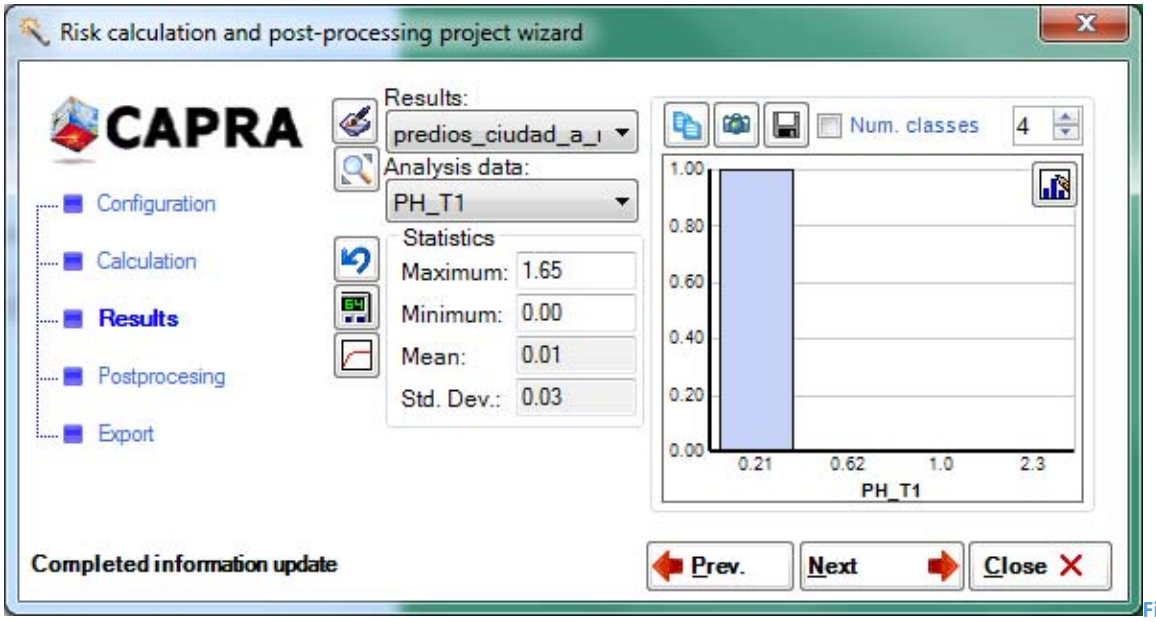


Figura 5-11 Resultado de CAPRA 2.0 para el CDU-2 Todos los escenarios. Dato analizado: PH-T1.

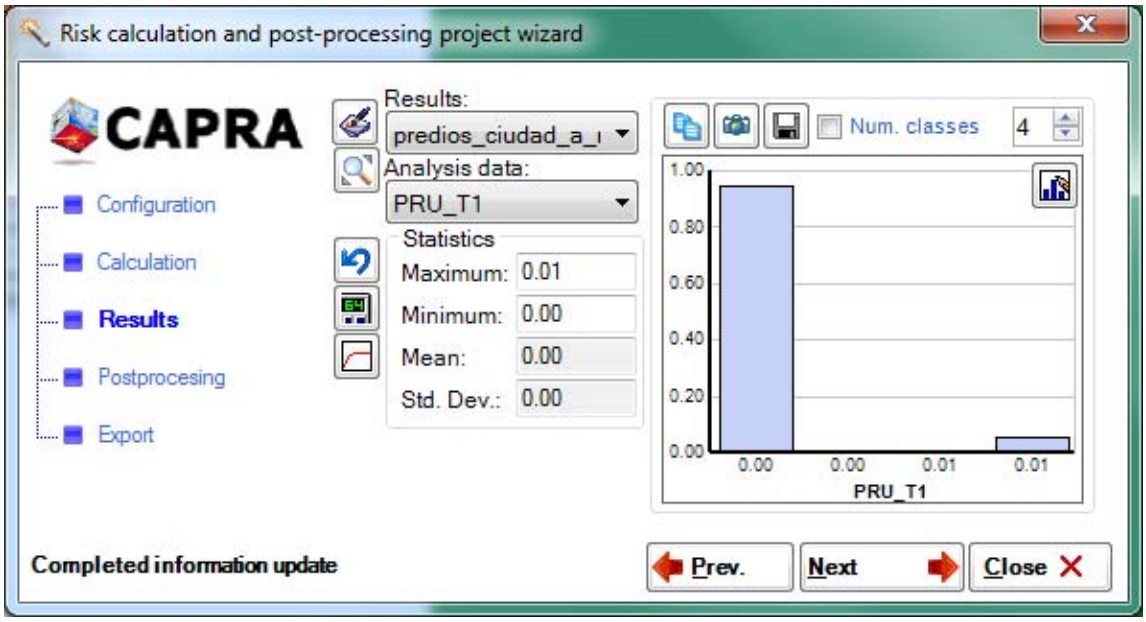


Figura 5-12 Resultado de CAPRA 2.0 para el CDU-2 Todos los escenarios. Dato analizado: PRU-T1.

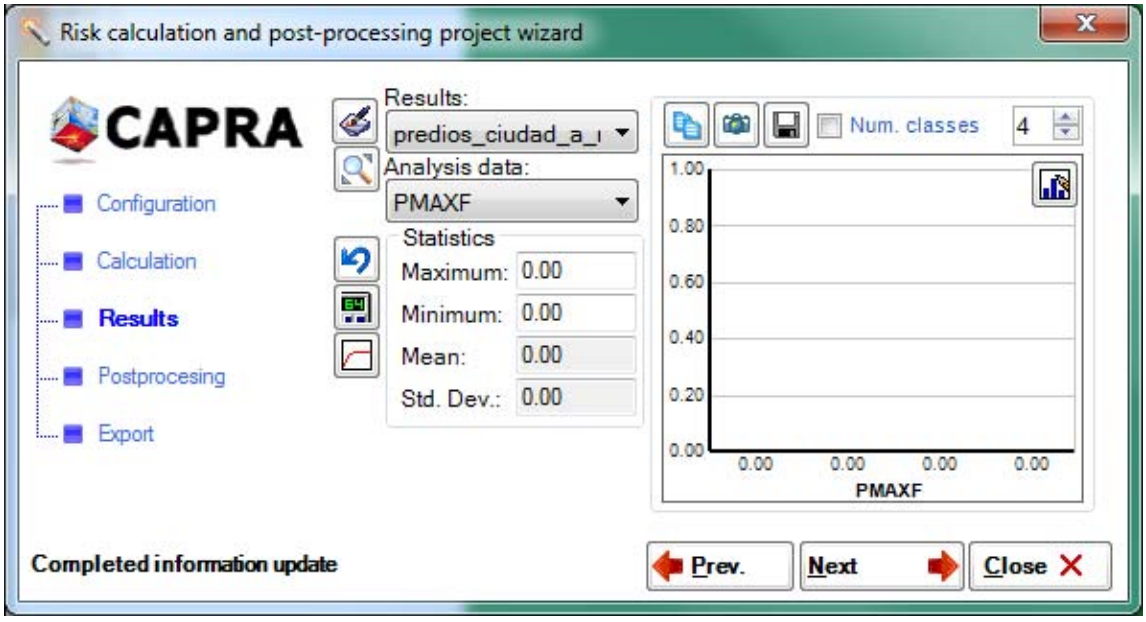


Figura 5-13 Resultado de CAPRA 2.0 para el CDU-2 Todos los escenarios. Dato analizado: PMAXF.

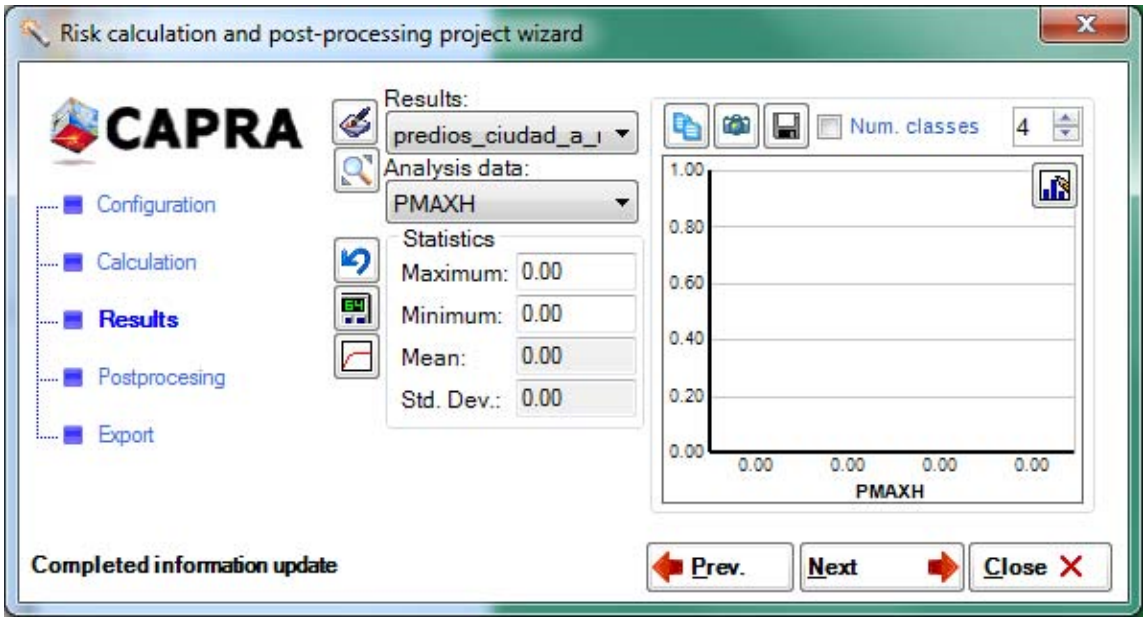


Figura 5-14 Resultado de CAPRA 2.0 para el CDU-2 Todos los escenarios. Dato analizado: PMAXH.

La línea base funcional está conformada por la imágenes anteriores que sintetizan los resultados de la ejecución de ambos casos de usos.

IMPLEMENTACIÓN DE LA ARQUITECTURA Y CODIFICACIÓN

A continuación se describe una serie de pasos realizados durante la implementación de la arquitectura propuesta, los cuales requieren como insumo el diseño de arquitectura conformado por los diagramas del modelado 4+1 desarrollados en el capítulo anterior.

Estos pasos no son la única forma de implementar el diseño de la arquitectura; sin embargo, se proponen a manera de procedimiento para replicar el resultado obtenido.

Proceso para reestructuración de código en capas y entidades de negocio

La arquitectura a implementar se centra en un diseño orientado al dominio. Esto se traduce en construir un modelo de dominio que contenga las clases y atributos que representen a las constantes, variables, términos y conceptos utilizados en el argot de la evaluación de riesgos naturales.

Las clases que forman el modelo de domino se les conoce como entidades de negocio, las cuales tienen por objetivo ser los contenedores de datos que moverán la información entre las tres capas: capa de datos, capa experta o de negocio, y capa de presentación o de GUI.

La versión 2.0 de CAPRA no cuenta con un modelo de dominio explícito, aunque sí cuenta con clases que agrupan atributos altamente cohesionados. Estas clases también tienen métodos diseñados para ofrecer autosuficiencia funcional como por ejemplo: calcularse, validarse, guardarse y llenarse a sí mismas, tal como se muestra en la figura siguiente.

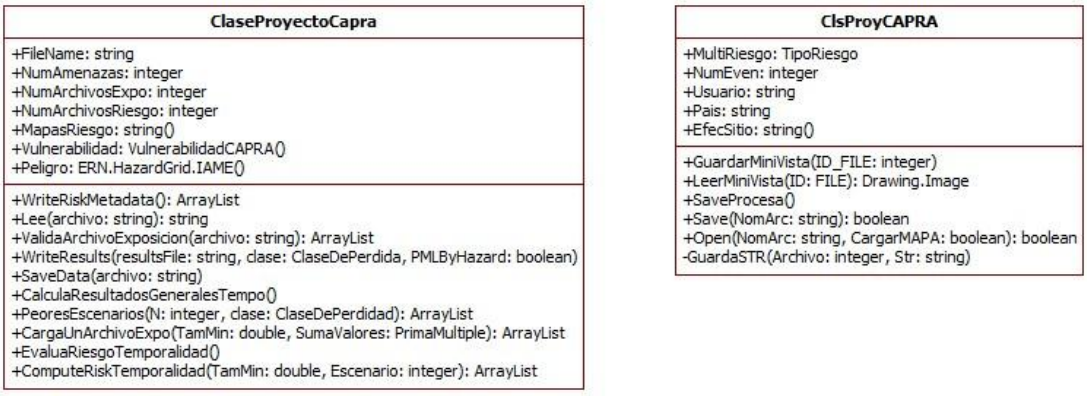


Figura 5-15 Ejemplo de clases autosuficientes de CAPRA 2.0 con métodos de acceso a datos y lógica de negocio. (Elaboración propia, 2014)

Dentro de la implementación de la arquitectura que trata este capítulo, se han creado clases de la capa de datos y capa experta, así como entidades de negocio a partir de dichas clases autosuficientes, utilizando para ello el proceso para reestructuración de código en capas y entidades de negocio que se presenta en la figura 5-16.

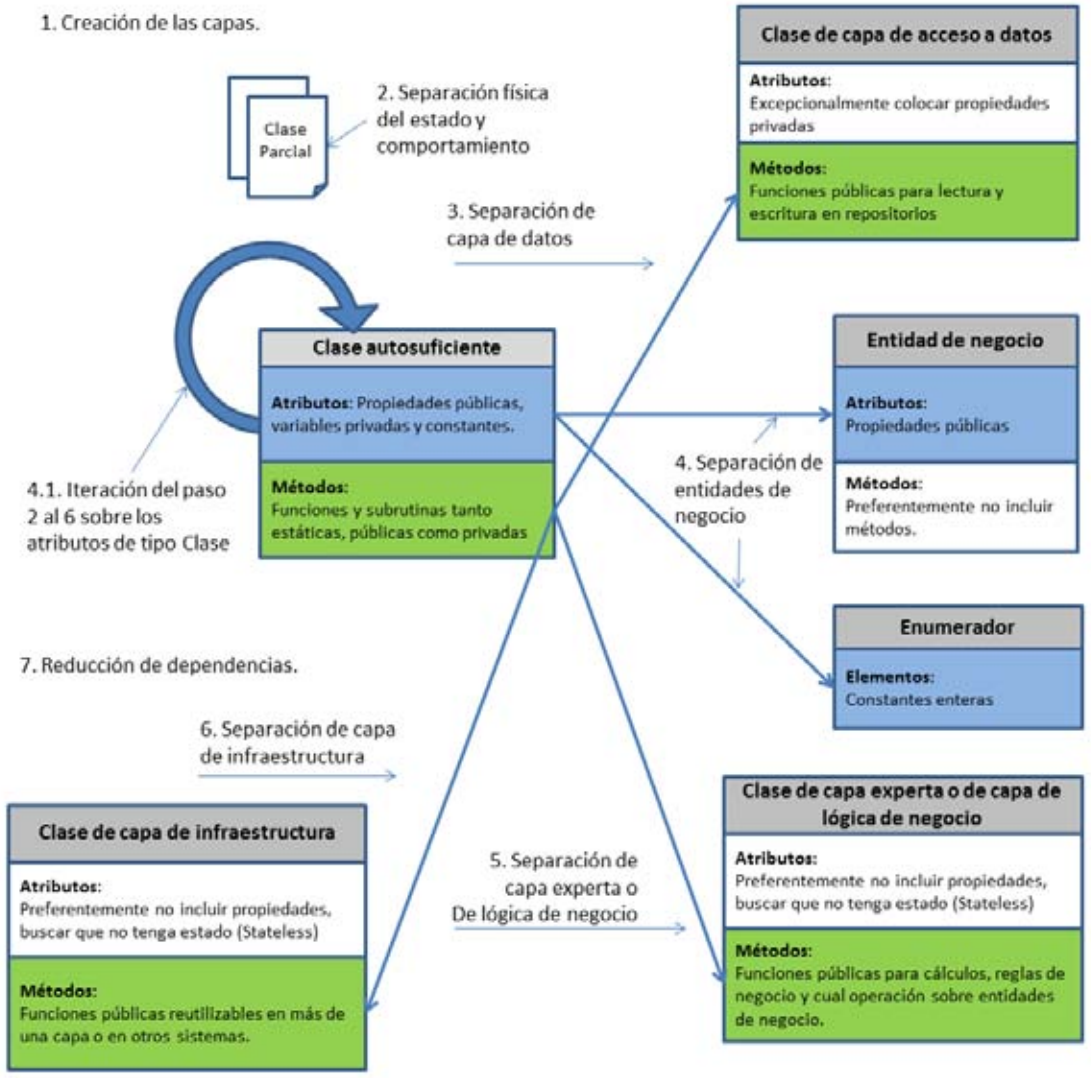


Figura 5-16 Proceso para reestructuración de código en capas y entidades de negocio (Elaboración propia, 2014)

Este proceso de reestructuración del código en capas y entidades de negocio se basa en 7 pasos descritos a continuación y se proponen como mecanismo complementario dentro de la implementación de la arquitectura.

1. **Creación de las capas.** El primer paso es crear la estructura de capas a partir de la arquitectura diseñada. La estructura de la nueva arquitectura se indica en la figura 4.8, la cual incluye capa de datos, capa experta o capa de negocio y capa de presentación, así como una capa transversal para componentes de infraestructura. En dicha estructura, cada capa hospeda a uno o varios componentes; a su vez cada componente es el contenedor de clases, enumeradores e interfaces.
2. **Separación física del estado y comportamiento.** En este paso se propone separar físicamente el código de la clase autosuficiente en dos archivos. El primero, con el estado de la clase que puede estar conformado por constantes, variables privadas y propiedades públicas. El segundo archivo con todos los métodos públicos y privados. Lo anterior con la intención de convertir al primer archivo con atributos en una clase de entidad de negocio, y el segundo dejarlo vacío tras mover los métodos a las diferentes capas.
3. **Separación de la capa de datos.** En este paso se identifican uno a uno los métodos orientados a guardar y leer información de un repositorio de datos, con la intención de moverlo a la capa de datos. Tanto la declaración del método como su código se deben modificar para recibir o entregar información mediante una entidad de negocio. Los métodos pueden ser renombrados empezando con un verbo en infinitivo que ofrezca claridad al propósito de su código. Internamente las referencias a la instancia de la clase autosuficiente – palabra reservada "Me" en Visual Basic.net - se reemplaza por una instancia de la entidad de negocio recibida o entregada según sea el caso.
4. **Separación de entidades de negocio.** Además de las entidades de negocio identificadas durante la separación de la capa de datos, también se debe identificar entidades exclusivas para la capa de negocio y su interacción con la capa de presentación. Las entidades de negocio pueden ser renombradas con sustantivos en singular que representen la cohesión y abstracción de sus atributos. Para aquellas constantes con valor numérico entero y para atributos con un conjunto limitado de valores a contener es conveniente la creación de enumerados.
   4.1. **Iteración sobre sus atributos de tipo clase autosuficiente.** Adicionalmente, para cada entidad de negocio se debe identificar sus atributos de tipo clase, lo cual implica la separación de una nueva entidad de negocio; convirtiendo así a este proceso en iterativo del paso 2 al 6.
5. **Separación de la capa experta o capa de lógica de negocio.** Los métodos restantes de la clase autosuficiente se deben mover a una nueva clase, cuyo nombre se oriente a un rol o servicio. Se debe evitar usar un sustantivo que se confunda con una entidad de negocio. Preferentemente, esta clase experta no debe tener propiedades que persistan su estado – debe ser stateless- de modo tal que sus métodos pueden ser invocados de manera independiente. Se define el método bajo el principio de recibir en sus parámetros toda la información requerida para la ejecución del mismo, y donde los parámetros principalmente corresponden a entidades de negocio.
6. **Separación de la capa de infraestructura.** Cuando se detectan métodos, clases e incluso bibliotecas completas que podrían ser utilizados en más de una capa, entonces, esos elementos son candidatos a trasladarse a la capa de infraestructura. Las clases y

componentes de esta capa se caracterizan por ser reutilizables inclusive en otros sistemas.

7. **Reducción de dependencias.** Después de reestructurar el código es posible que algunos componentes queden vacíos o se unifiquen. En tal caso, conviene eliminar sus proyectos de la solución y eliminar las referencias a dichos componentes.

Resultado de la creación de las capas

Como primer paso se crea una solución de Visual Studio 2010 con nombre "CAPRAGIS con BibliotecaClases.sln". Esta solución debe incluir el código fuente de los 20 proyectos que conforman CAPRA, los cuales están plasmados en el diagrama de componentes del sistema CAPRA v2.0 (Figura 4-2).

Después se agregan los nuevos proyectos indicados en el diagrama de capas y espacios de nombre (Figura 4-8). Estos proyectos corresponden a los componentes de la arquitectura en capas propuesta, así como a un componente para las clases del modelo de dominio.

La siguiente tabla indica los proyectos creados y sus propiedades principales.

Componente	Proyecto	Namespace default
CAPRAv3.Entities.dll	V3_Entidades.CAPRA.vbproj	Entidades.CAPRA
CAPRAv3.DataAccess.dll	V3_CAPRA.DataAccess.vbproj	DataAccessLayer.CAPRA
CAPRAv3.ExpertServices.dll	V3_CAPRA.ExpertServices.vbproj	ExpertLayer.CAPRAServices
Infraestructure.ERN. RandomFunctions.dll	V3_Infraestructure.ERN. RandomFunctions.vbproj	Infraestructure.ERN. RandomFunctions
Infraestructure.Tools.dll	V3_Infrastructure.Tools.vbproj	Infraestructure.Tools

Tabla 5-2 Proyectos nuevos agregados a la solución de Visual Studio. (Elaboración propia, 2014)

Resultado de la separación física del estado y comportamiento

En el caso de Visual Studio.net la separación física se realizó mediante el uso de dos clases parciales almacenadas en archivos físicos distintos.

Adicionalmente si el archivo a separar contenía otras clases entonces se separaba cada una de ellas en archivos independientes. Un ejemplo de ello es el archivo "Perdidas.vb" cuya separación se muestra en la figura 5-17.

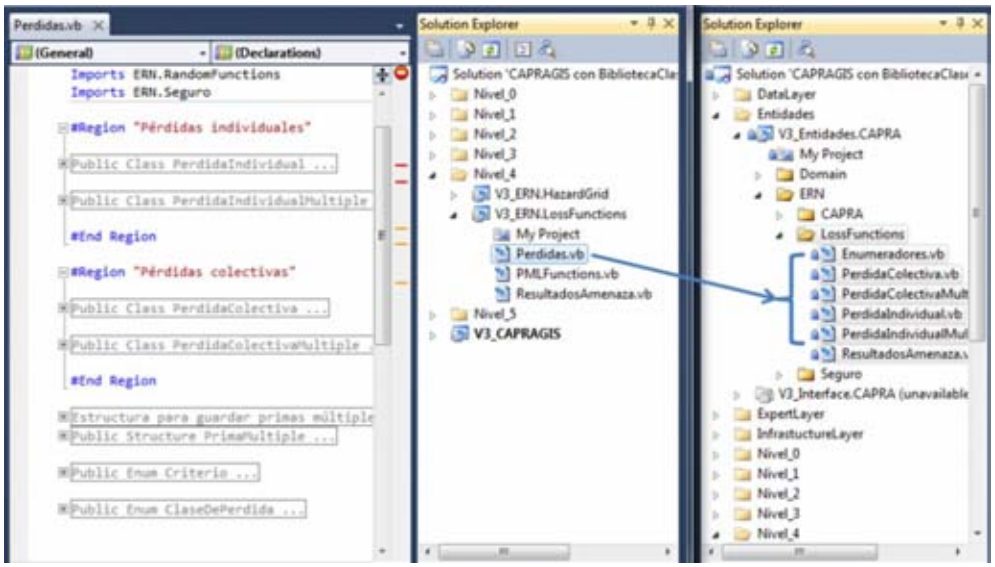


Figura 5-17 Ejemplo de separación física del archivo Perdidas.vb en varias clases. (Elaboración propia, 2014)

Resultado de la separación de capa de datos

La figura siguiente muestra los métodos y clases resultantes de la separación de la capa de datos tras la aplicación del proceso de reestructuración de código.

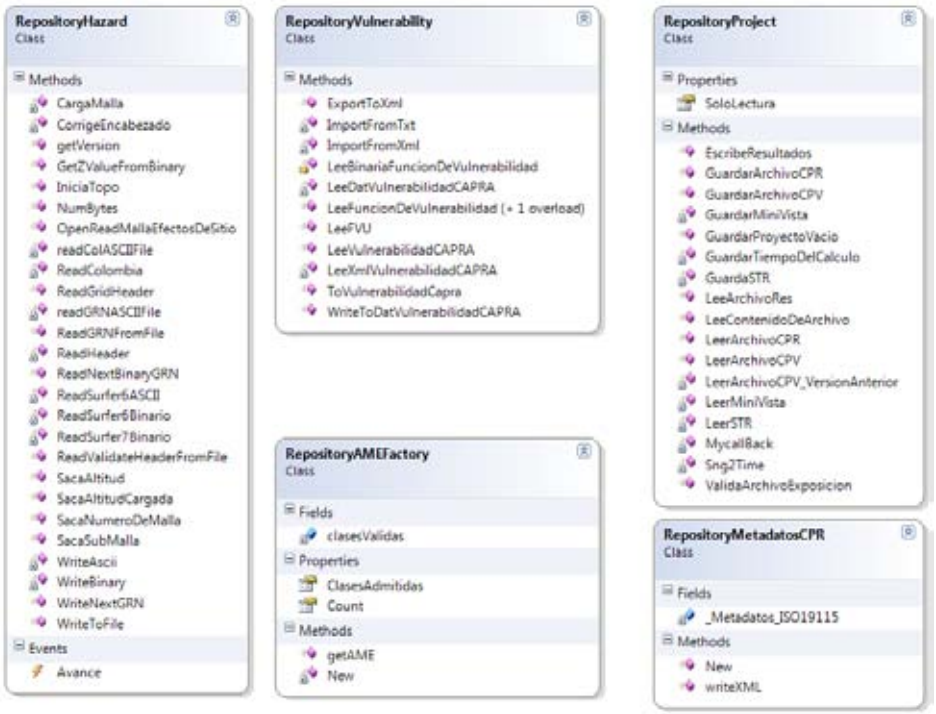


Figura 5-18 Clases y métodos que componen la capa de datos de CAPRA 3.0. (Elaboración propia, 2014)

Resultado de la separación de entidades de negocio

La figura 5-21 muestra el resultado de la separación de entidades de negocio. Todas estas clases forman el modelo de dominio y están contenidas bajo un único namespace EntityLayer.CAPRA y un solo componente CAPRAv3.Entity.dll. Aunque dentro de la solución de Visual Studio, las clases se encuentran organizadas en subcarpetas físicas como se observa en la figura 5-19.

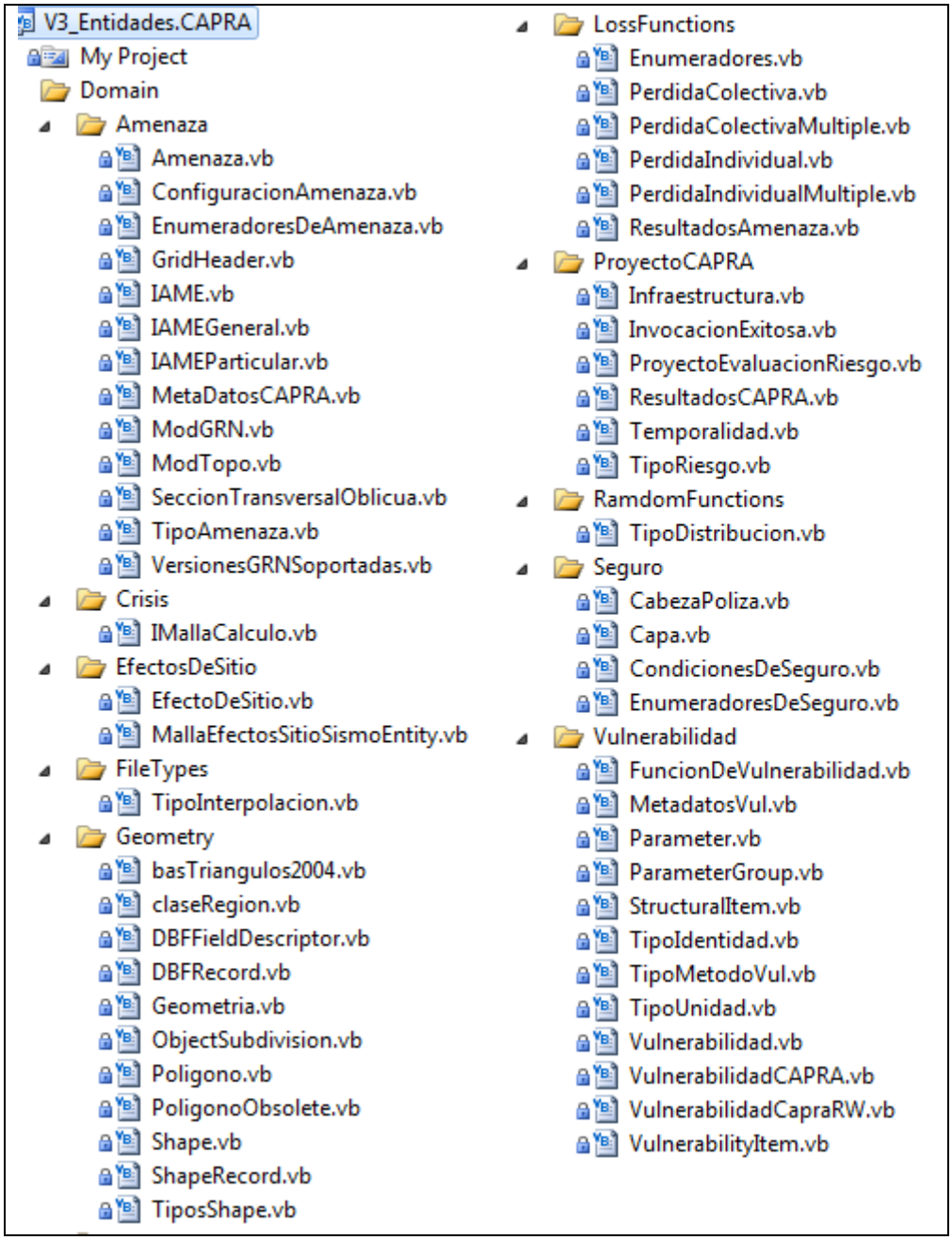


Figura 5-19 Clases del modelo de domino: Entidades de negocio de CAPRA 3.0. (Elaboración propia, 2014)

Cabe mencionar que las clases del modelo de dominio también podrían dividirse en más de un proyecto y/o espacio de nombre, en el caso que fuera necesario para su reutilización segmentada en otros sistemas.

Resultado de la separación de la capa experta o capa de lógica de negocio

Las nuevas clases de la capa experta resultantes de la ejecución del proceso de reestructuración de código se muestran en el siguiente diagrama de clases.

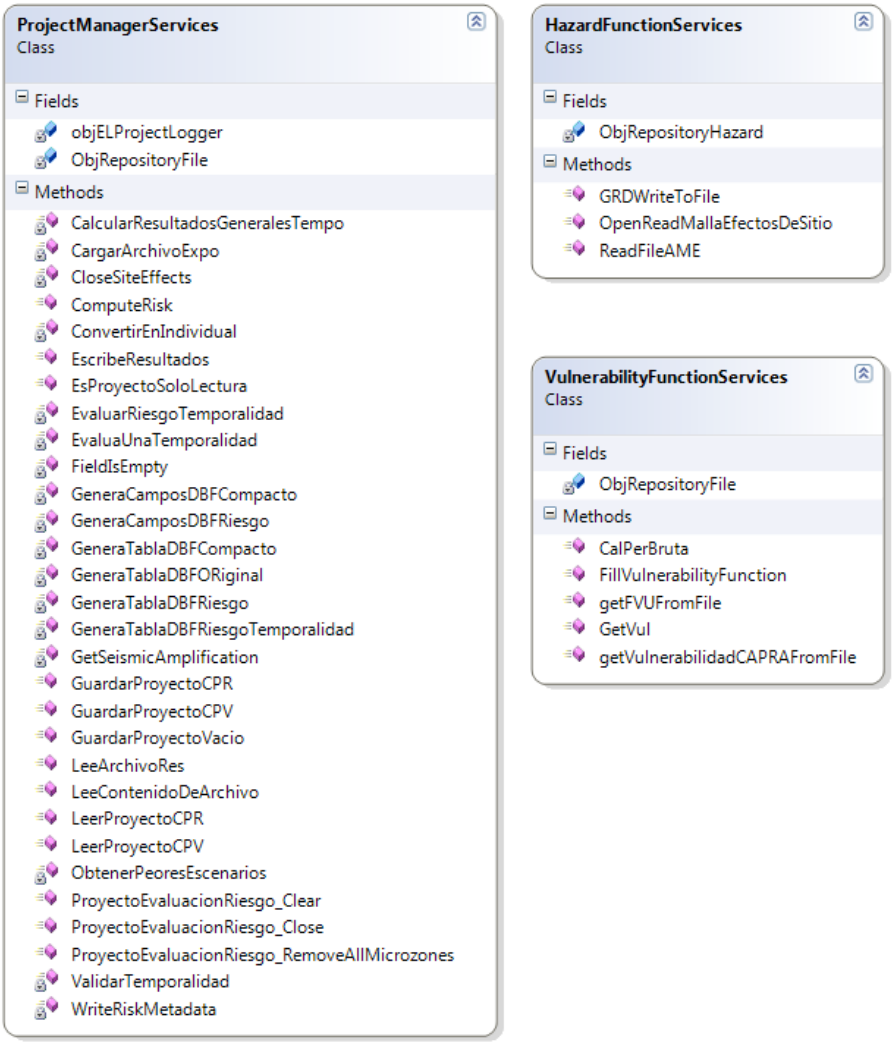


Figura 5-20 Clases de la capa experta o capa de lógica de negocio de CAPRA 3.0. (Elaboración propia, 2014)

Todos los métodos de las clases de la capa experta o capa de negocio se implementaron sin estado; por lo tanto en Visual Basic.net se declararon como funciones tipo Shared; conocido en otros lenguajes como funciones estáticas.

Resultado de la separación de la capa de infraestructura

La capa de infraestructura se compone de dos bibliotecas: Infraestructure.Tools.dll e Infraestructure.ERN.RandomFunctions.dll. La segunda originalmente llamada ERN.RandomFunctions.dll. Por su lado, Infraestructura.Tools.dll contiene clases y métodos identificados como generales e incluso reutilizables para otros sistemas fuera del dominio de la evaluación de riesgos naturales. El diagrama de clases de este componente se muestra en la figura 5-21.

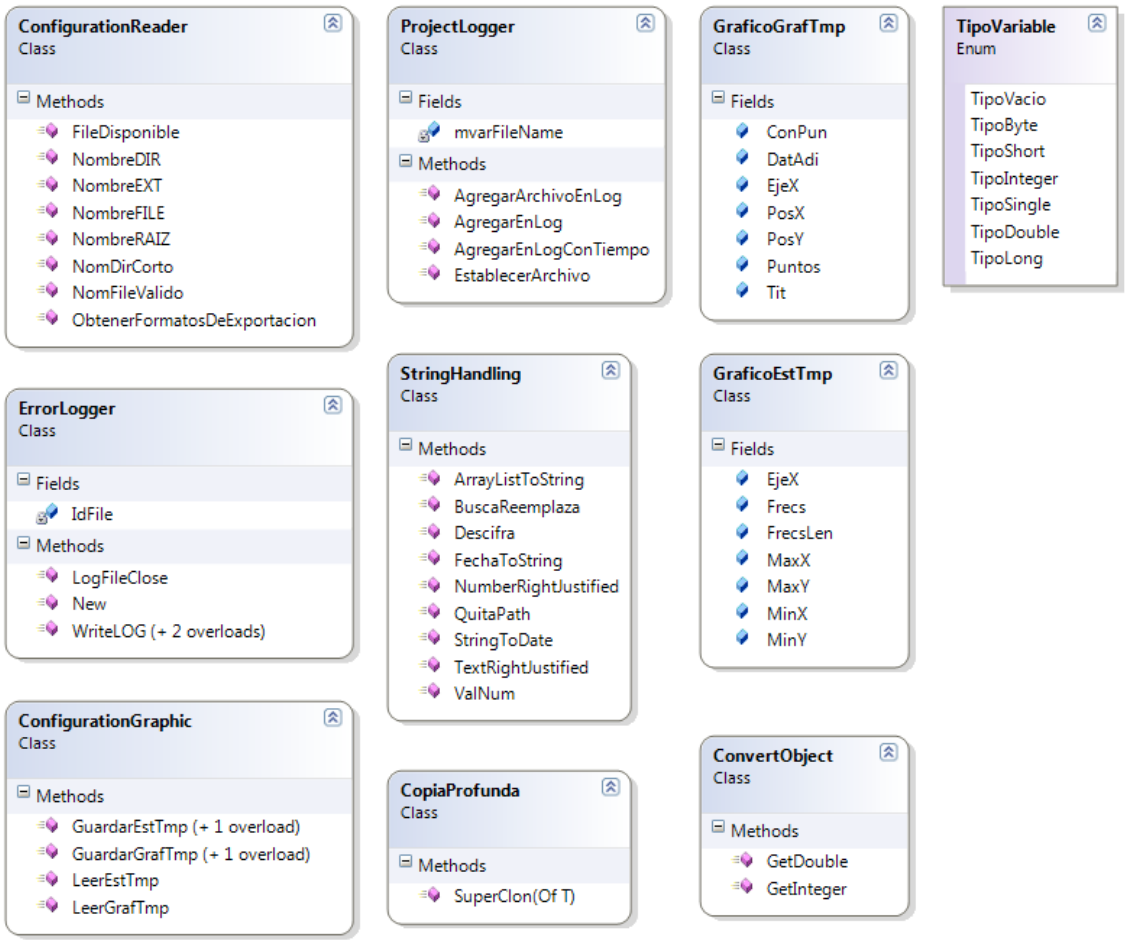


Figura 5-21 Diagrama de clases de la capa de infraestructura de CAPRA 3.0. (Elaboración propia, 2014)

Resultado de la reducción de dependencias

Como resultado de la reestructuración de código en capas y entidades de negocio se separaron los elementos de los 7 componentes mostrados en la figura 5-22 y sus proyectos se eliminaron de la solución, lo anterior dio paso a la reducción de dependencias.

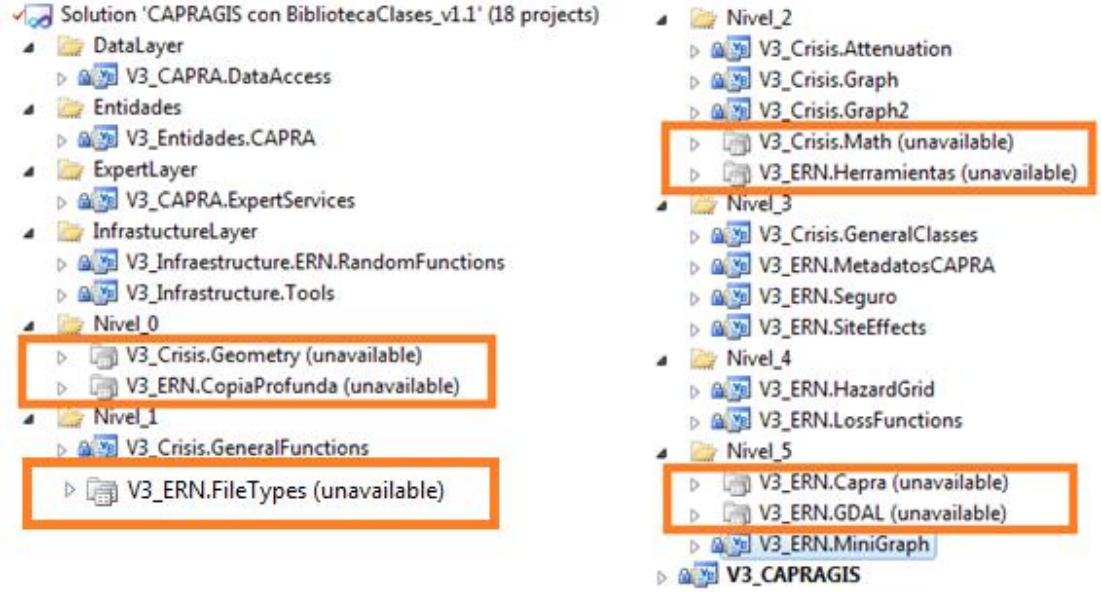


Figura 5-22 Componentes eliminados en CAPRA 3.0 derivado de la separación en capas y entidades de negocio. (Elaboración propia, 2014)

Como parte de la reducción de dependencias se procedió a depurar cada uno de los proyectos para eliminar las referencias sin uso mediante la opción respectiva en Visual Studio.net.

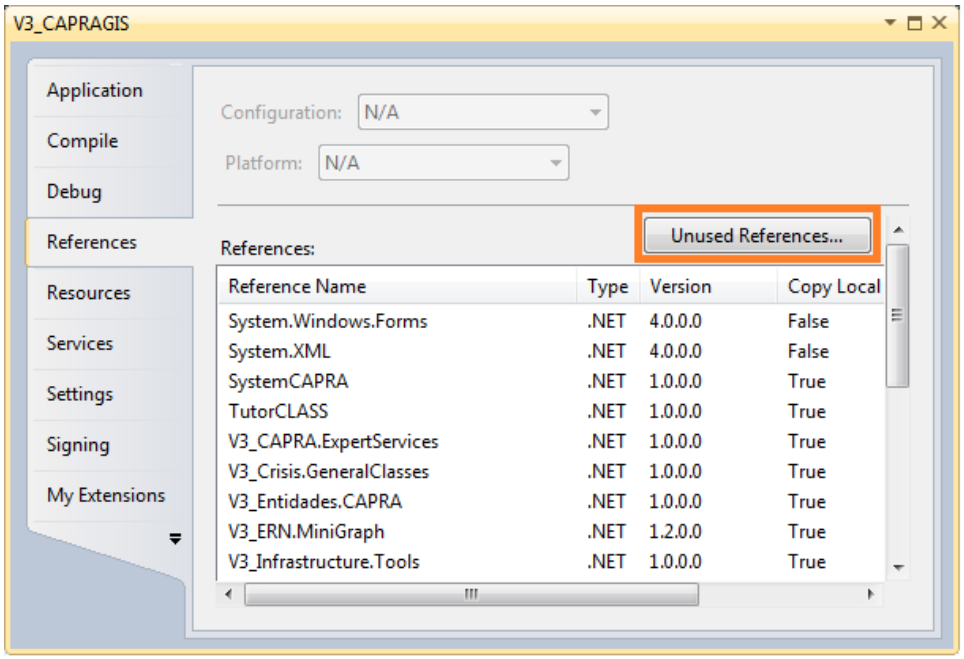


Figura 5-23 Opción de Visual Studio.net para eliminar referencias sin uso.

Como resultado de la depuración de componentes reestructurados y dependencias eliminadas, a continuación se compara el diagrama de componentes original de la versión CAPRA 2.0 (presentado en la figura 4-2) contra el mismo diagrama tras la depuración.

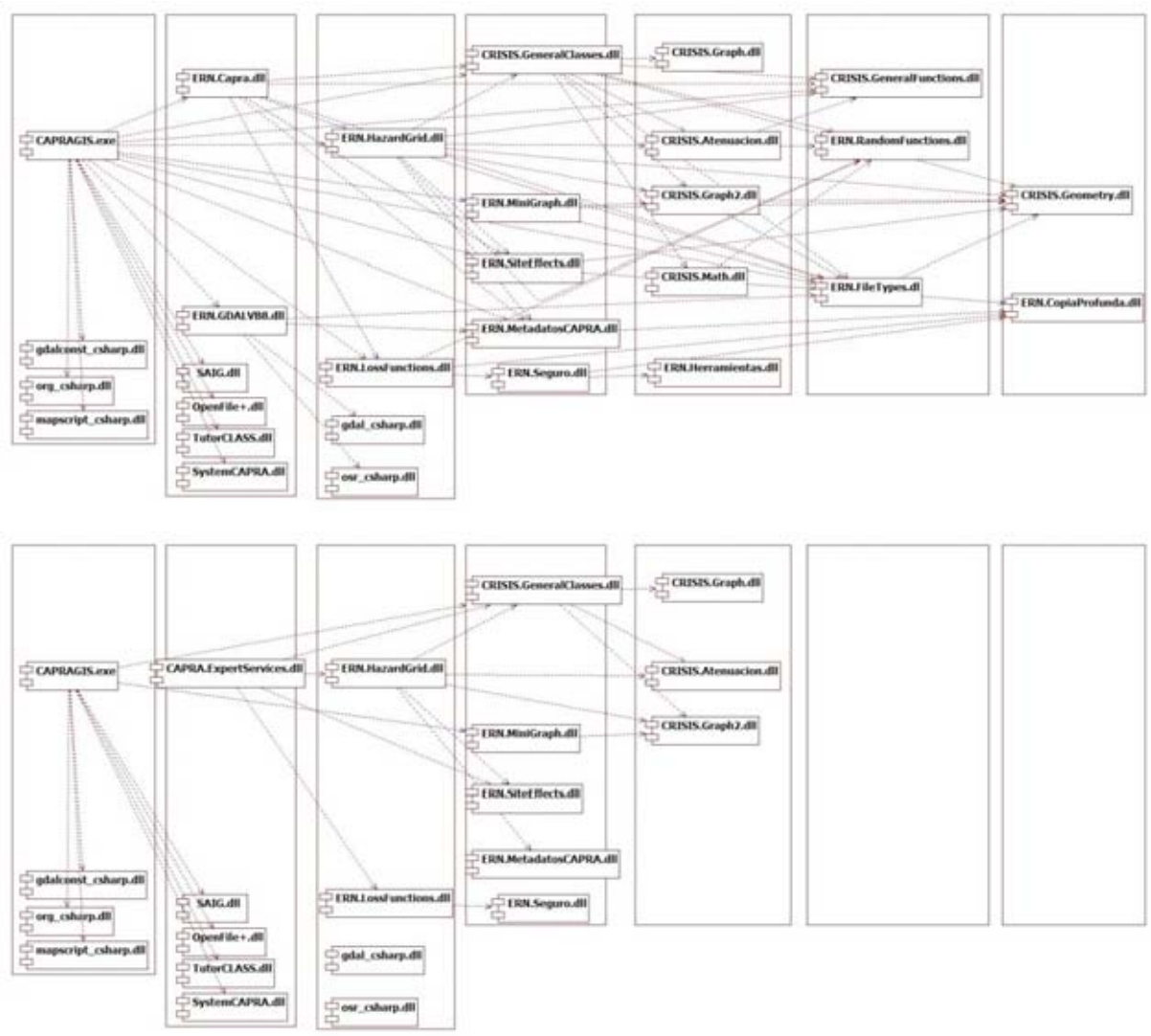


Figura 5-24 Resultado de depuración de componentes y simplificación de dependencias entre la capa de presentación y la capa de negocio. El antes se muestra en el diagrama superior y el después en diagrama inferior.

El diagrama inferior contenido en la figura 5-24 muestra el resultado de la depuración de componentes y simplificación de dependencias, el cual es solamente la unión de dos diagramas de componentes presentados en las figuras 4-9 y 4-10 que corresponden a la capa de presentación y capa experta respectivamente.

En la figura siguiente se muestra de lado izquierdo las 10 dependencias que tiene el componente de la GUI de la versión CAPRA 2.0, en tanto de lado derecho se muestra que para el mismo componente GUI de la versión 3.0 se logró reducir su dependencia a solamente 2 de los 10 de componentes originales.

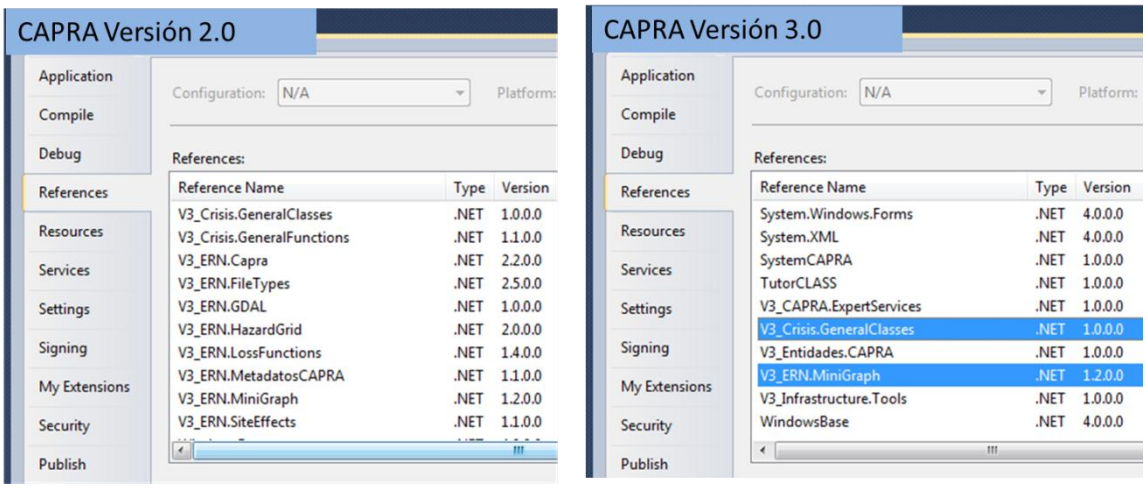


Figura 5-25 Simplificación de dependencias de la capa de presentación GUI hacia otros componentes.

Las tres nuevas dependencias obedecen a la implementación de la arquitectura propuesta donde V3_CAPRA.ExpertServices, V3_Entidades.CAPRA y V3_Infrastructure.Tools corresponden a servicios de la capa experta, entidades del modelo de dominio y capa de infraestructura respectivamente.

Revisión de espacios de nombre

Los espacios de nombres en cada componente se apegaron al diseño de acuerdo al diagrama de capas y espacios de nombre mostrado en la figura 4-8.

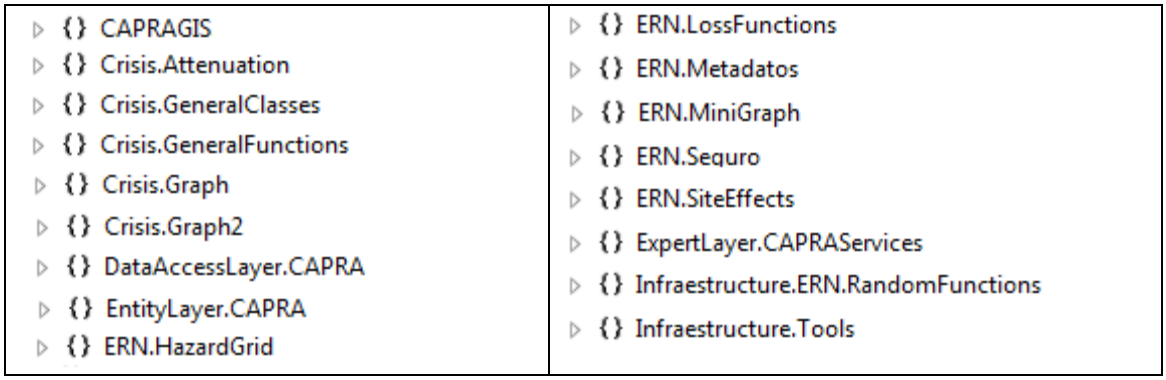


Figura 5-26 Espacios de nombre (namespace) implementados en CAPRA 3.0. (Elaboración propia, 2014)

Acotación de conflictos

Durante el proceso de reestructuración se encontró problemáticas en el código que impiden la reestructuración completa. Estos conflictos se acotan en la tabla siguiente.

Conflicto	Acotación	Consecuencia
Biblioteca SAIG	Varias pantallas (formularios) contenidos en el componente V3_CAPRAGIS de la capa de presentación dependen de la biblioteca SAIG.dll. A su vez la biblioteca depende de otras bibliotecas de CAPRA 2.0. Debido a que no se tiene el código fuente por tanto no es posible realizar modificaciones para eliminar dichas dependencias.	Para el correcto funcionamiento de la GUI es necesario mantener algunas bibliotecas compiladas de CAPRA 2.0
Interfaces expuestas	Algunas interfaces entre ellas IAME, IAMEGeneral, IAMEParticular se encontraban en una biblioteca que ha sido eliminada tras la reestructuración y con base en la inspección del código y los comentarios es presumible que dichas interfaces sean requeridas por otros sistemas.	Los sistemas que dependan de las interfaces reubicadas requieren necesariamente la actualización de referencias hacia los nuevos componentes creados en CAPRA 3.0
Métodos no utilizados	Existen métodos que no son invocados dentro de CAPRA por lo tanto es código en desuso absoluto o que es invocado únicamente por otras aplicaciones de manera externa.	Para fines de este trabajo se consideró a dichos métodos en desuso absoluto por lo tanto no se aplicó reestructuración de código en ellos.

Tabla 5-3 Conflictos encontrados durante la reestructuración del código. (Elaboración propia, 2014)

COMPROBACIÓN DE LA FUNCIONALIDAD

En este paso se procede a comprobar que la nueva versión 3.0 del sistema CAPRA cumple con la misma funcionalidad que la versión anterior. Para lograrlo se debe ejecutar los dos casos de uso definidos como línea base funcional.

La comprobación consiste en comparar los resultados mostrados en cada una de las pantallas de las figuras 5-1 a la 5-14.

Resultados del escenario 1: CDU-1 Cálculo del riesgo sísmico para un único escenario

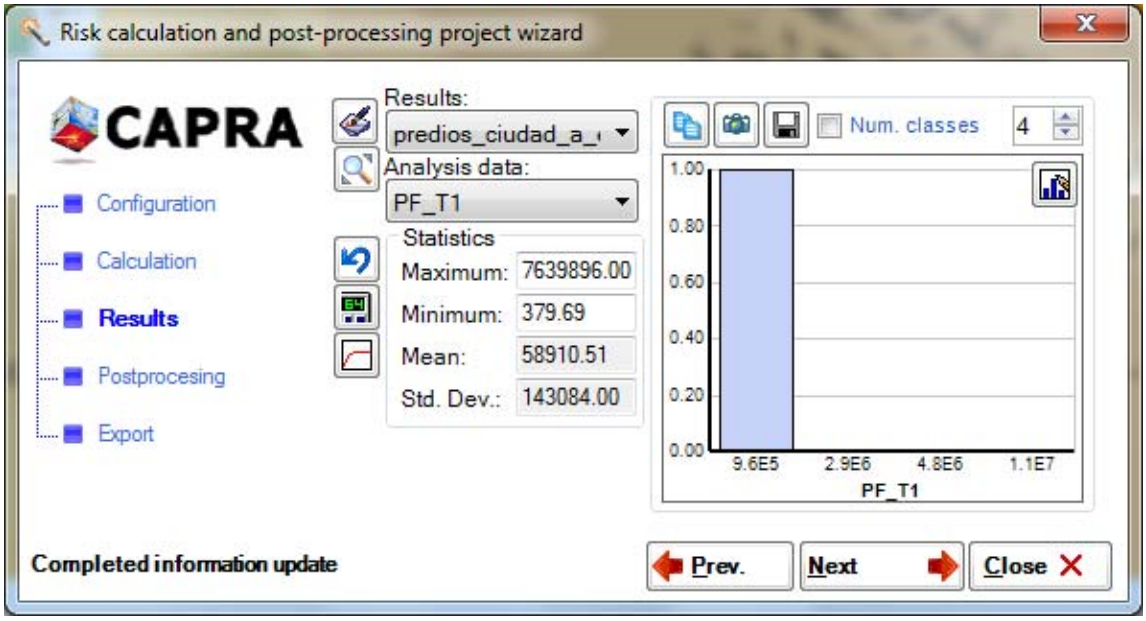


Figura 5-27 Resultado de CAPRA 3.0 para el CDU-1 Escenario 407. Dato analizado: PF_T1.

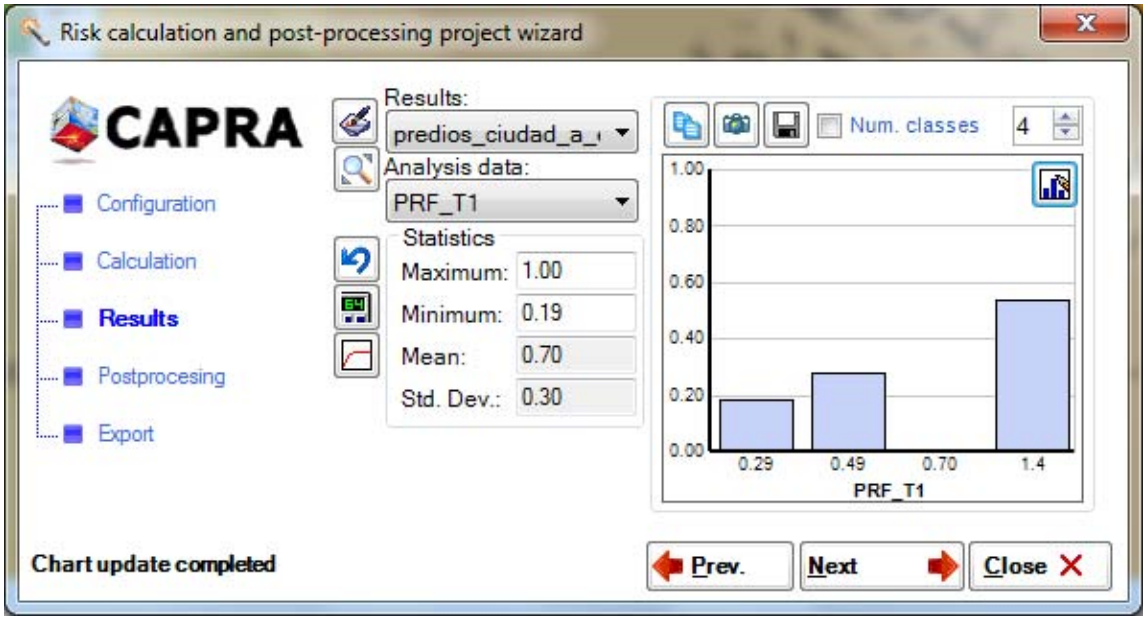


Figura 5-28 Resultado de CAPRA 3.0 para el CDU-1 Escenario 407. Dato analizado: PRF_T1.

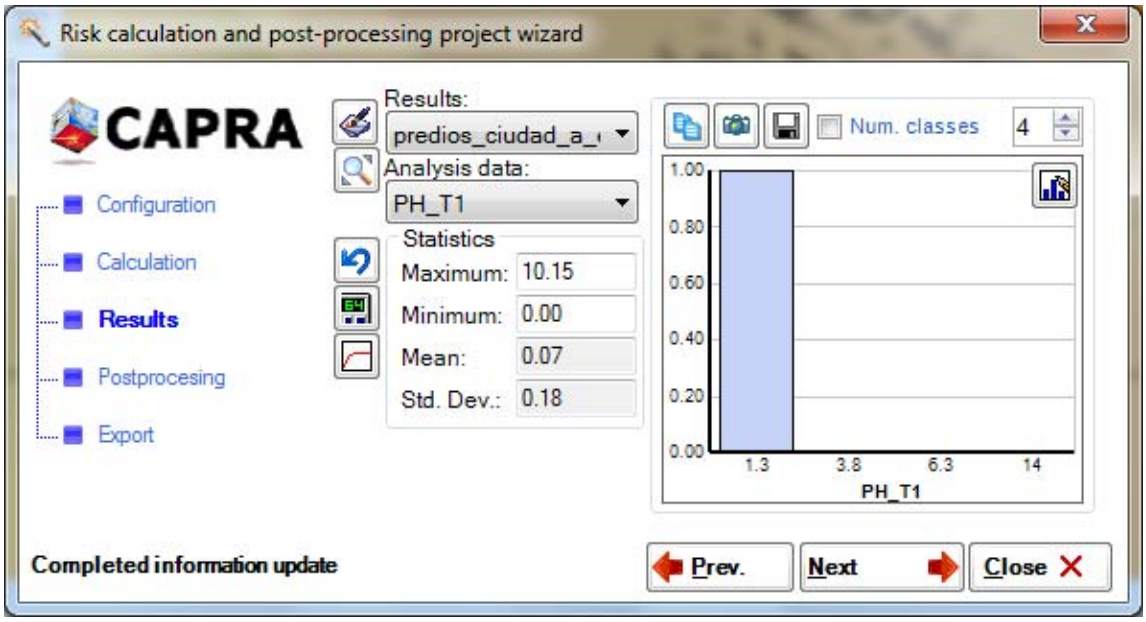


Figura 5-29 Resultado de CAPRA 3.0 para el CDU-1 Escenario 407. Dato analizado: PH_T1.

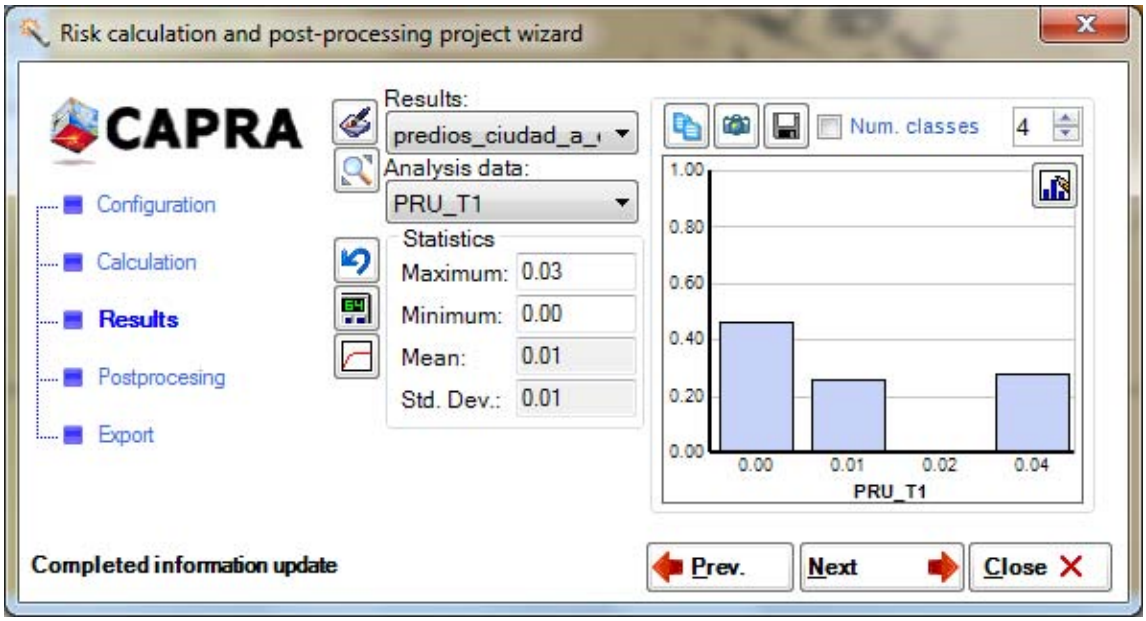


Figura 5-30 Resultado de CAPRA 3.0 para el CDU-1 Escenario 407. Dato analizado: PRU_T1.

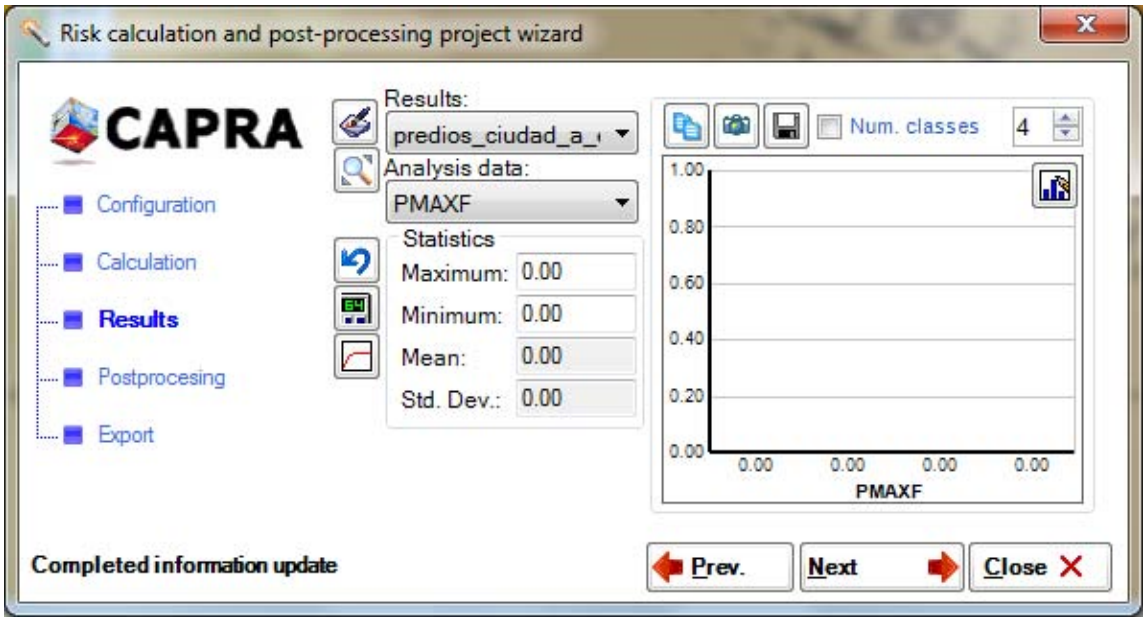


Figura 5-31 Resultado de CAPRA 3.0 para el CDU-1 Escenario 407. Dato analizado: PMAXF.

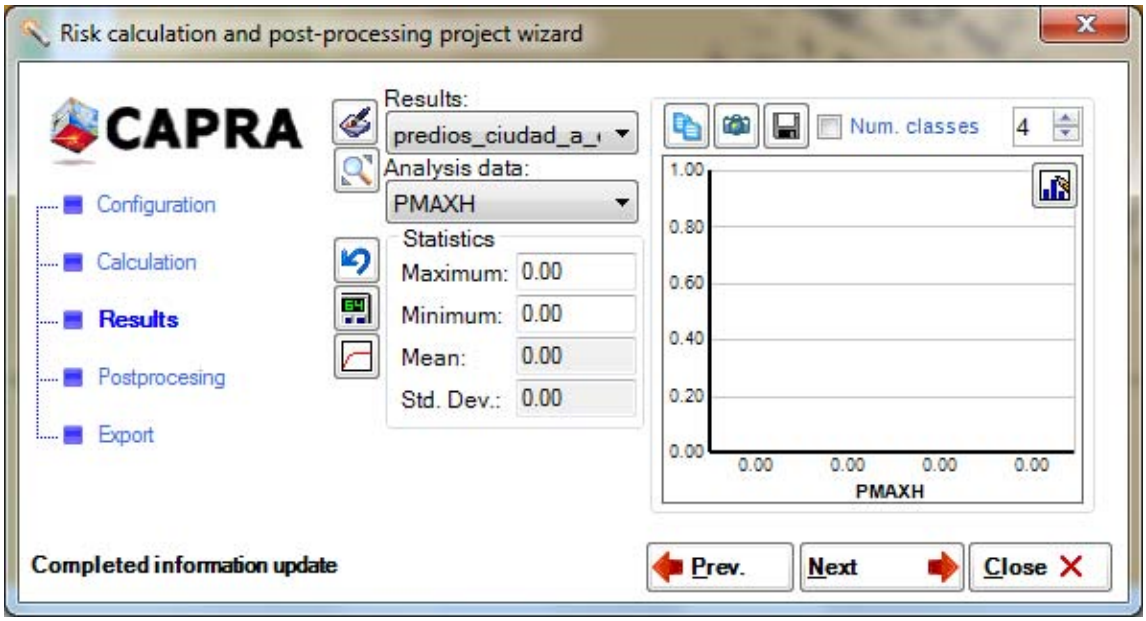


Figura 5-32 Resultado de CAPRA 3.0 para el CDU-1 Escenario 407. Dato analizado: PMAXH.

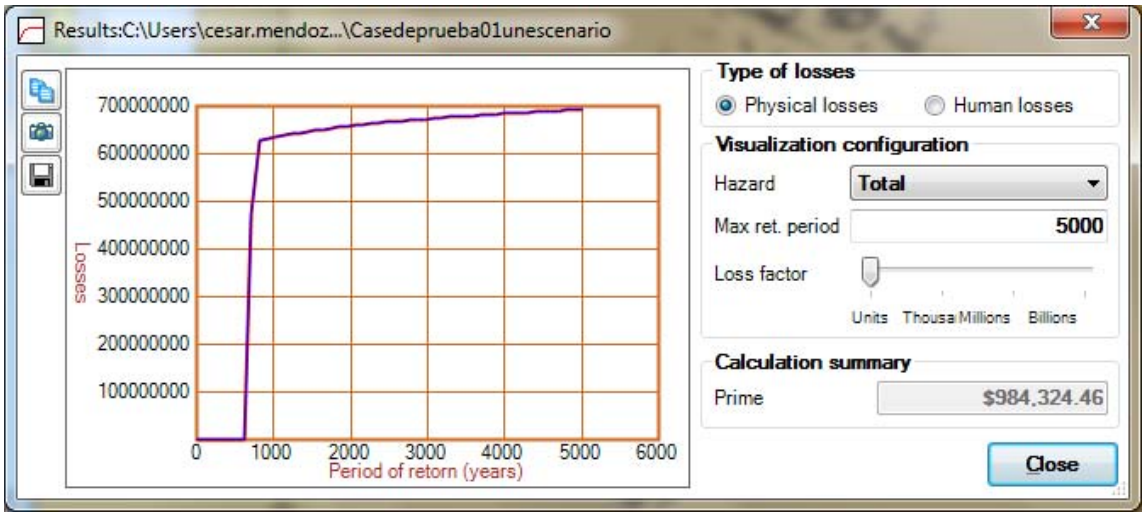


Figura 5-33 Resultado de CAPRA 3.0 para el CDU-1 Escenario 407. Dato analizado: Pérdidas físicas.

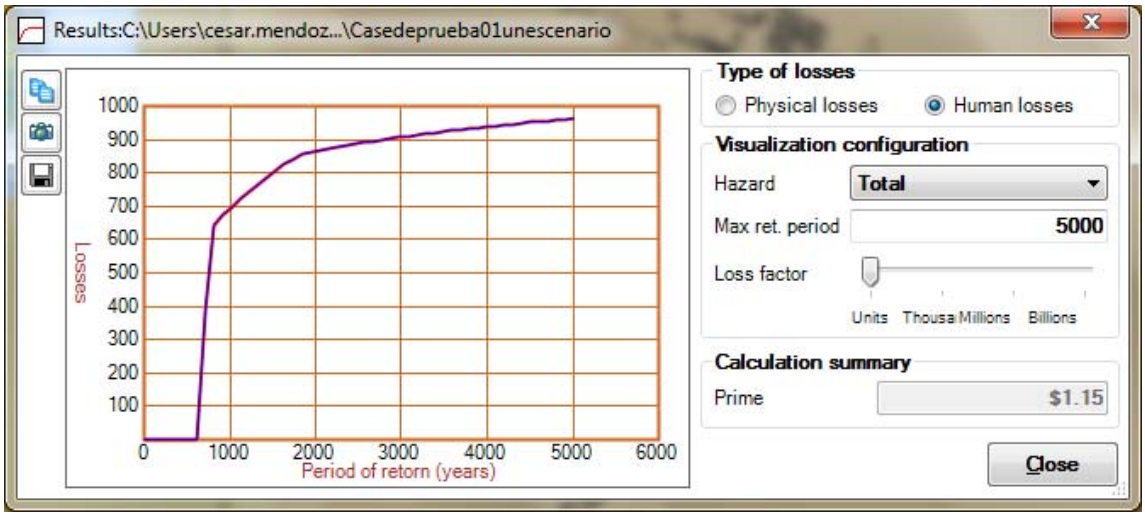


Figura 5-34 Resultado de CAPRA 3.0 para el CDU-1 Escenario 407. Dato analizado: Pérdidas humanas.

Con las imágenes anteriores que conforman el resultado de la ejecución del caso de uso 01 utilizando la nueva versión 3.0 de CAPRA queda comprobada su funcionalidad idéntica a la versión 2.0

Resultados del escenario 2: CDU-2 Cálculo del riesgo sísmico para un conjunto de riesgo estocástico

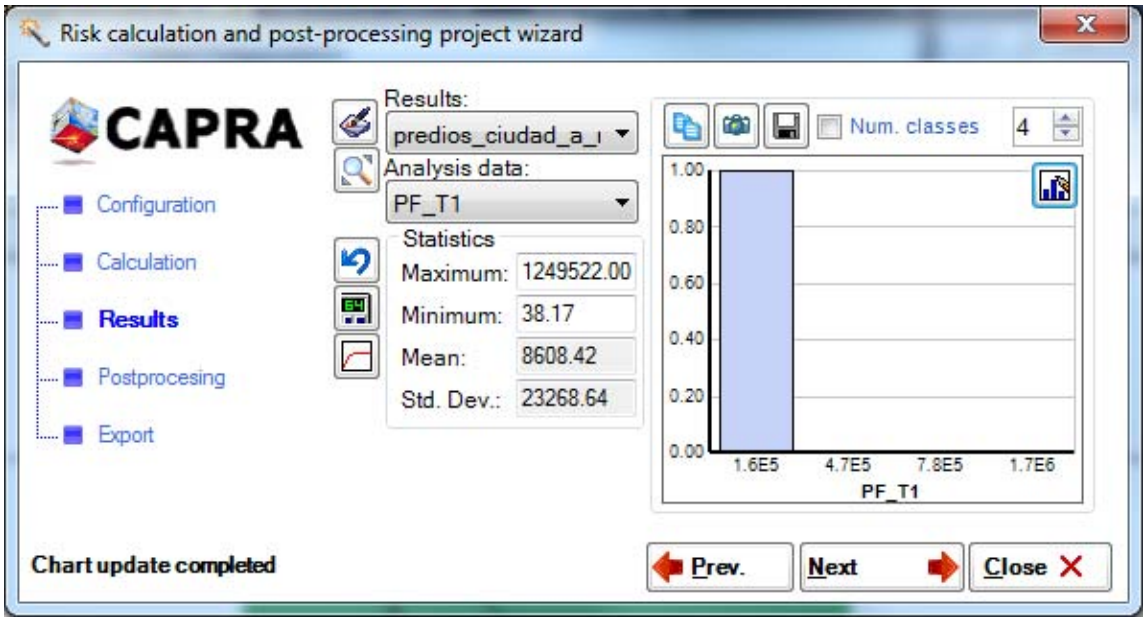


Figura 5-35 Resultado de CAPRA 3.0 para el CDU-2 Todos los escenarios. Dato analizado: PF-T1.

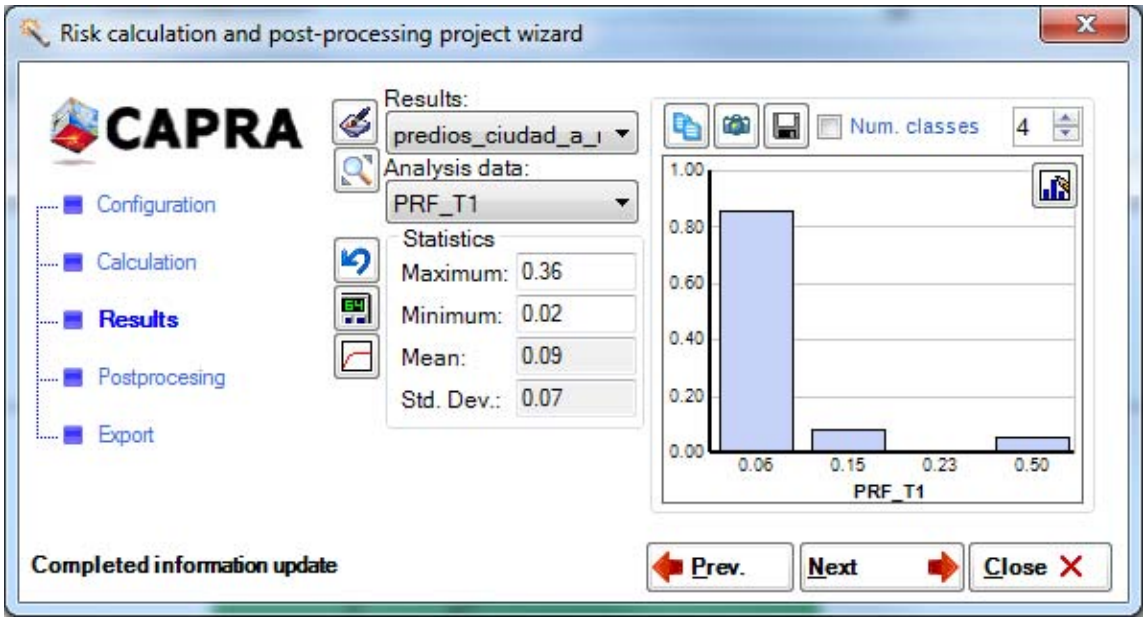


Figura 5-36 Resultado de CAPRA 3.0 para el CDU-2 Todos los escenarios. Dato analizado: PRF-T1.

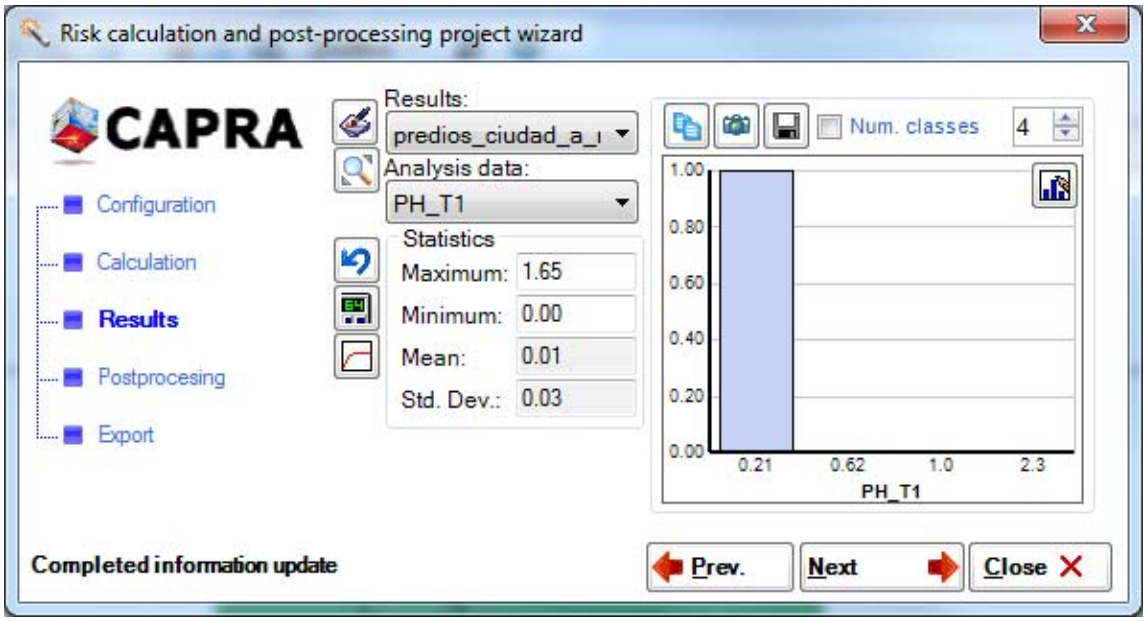


Figura 5-37 Resultado de CAPRA 3.0 para el CDU-2 Todos los escenarios. Dato analizado: PH-T1.

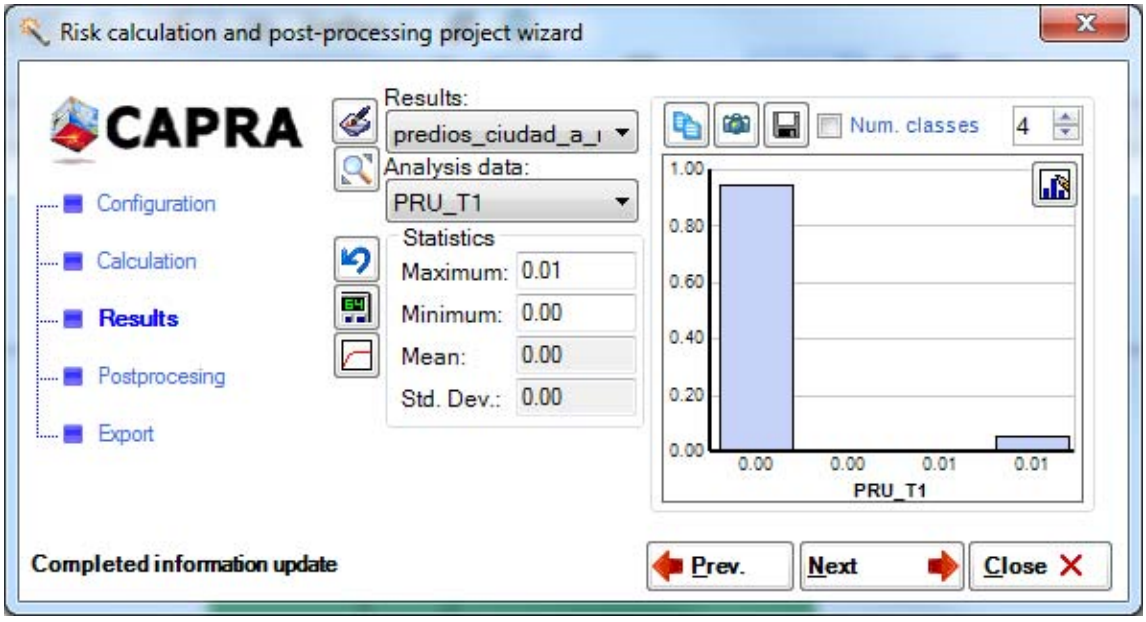


Figura 5-38 Resultado de CAPRA 3.0 para el CDU-2 Todos los escenarios. Dato analizado: PRU-T1.

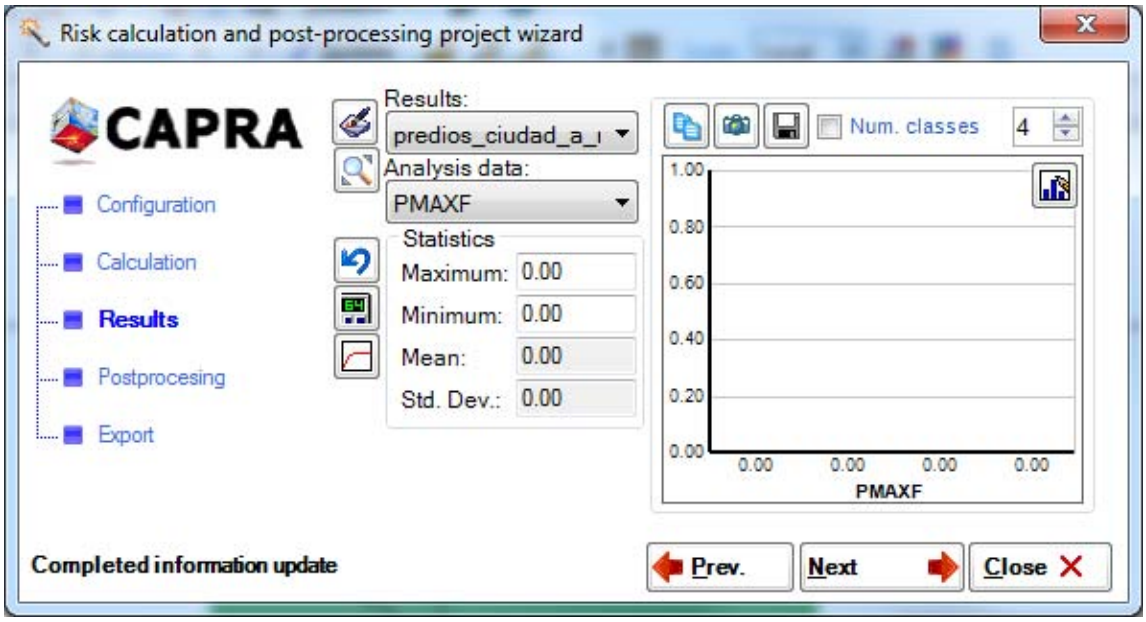


Figura 5-39 Resultado de CAPRA 3.0 para el CDU-2 Todos los escenarios. Dato analizado: PMAXF.

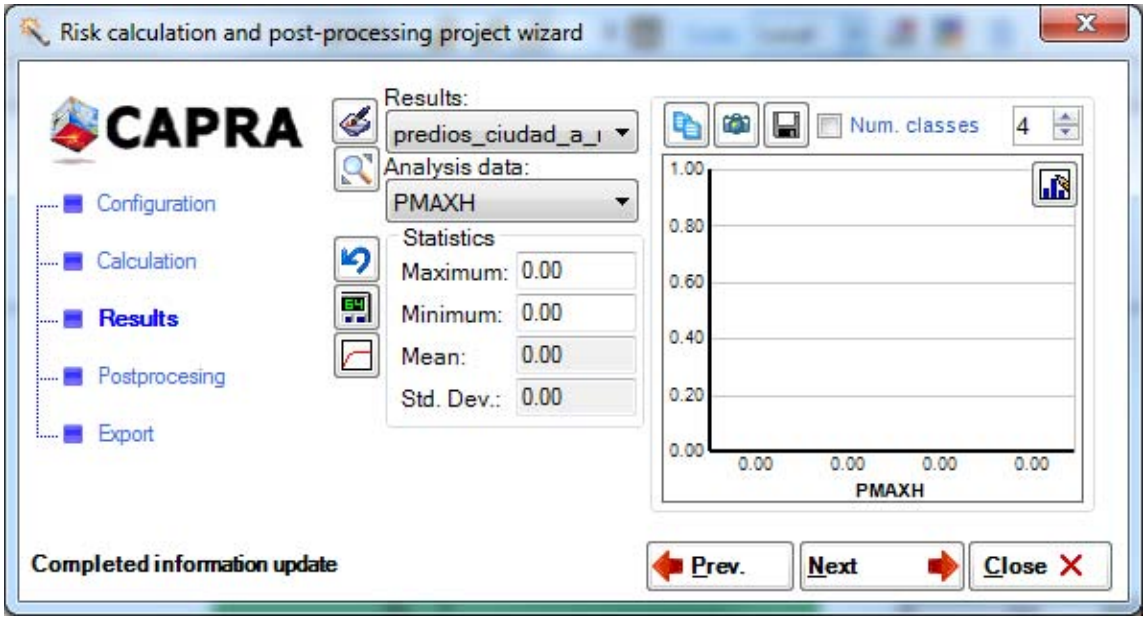


Figura 5-40 Resultado de CAPRA 3.0 para el CDU-2 Todos los escenarios. Dato analizado: PMAXH.

Se comparó los resultados del segundo escenario CDU-2 - figuras 5-35 a la 5-40 - contra los resultados de la línea base funcional –figura 5-9 a la 5-14- donde se observan los valores idénticos, por lo tanto, con esto queda comprobado que CAPRA versión 3.0 soporta la misma funcionalidad de negocio de la versión 2.0.

ACTUALIZACIÓN DE DOCUMENTACIÓN DE DISEÑO

La ingeniería de software dicta como buena práctica la actualización de documentación previa, en este caso durante la implementación de arquitectura se actualizó algunos de los diagramas de la arquitectura diseñada con el modelo 4+1.

Dichos cambios se aplicaron al contenido del capítulo 4 previo a la publicación de este trabajo; sin embargo es oportuno enfatizar la realización de este paso para quien considere replicar los pasos de esta implementación.

En el capítulo 4 se comenta que la vista lógica del modelo 4+1 era una versión preliminar donde la versión completa se obtiene al finalizar la implementación. Por ende, a continuación se presenta la versión actualizada del modelo de dominio de la vista lógica.

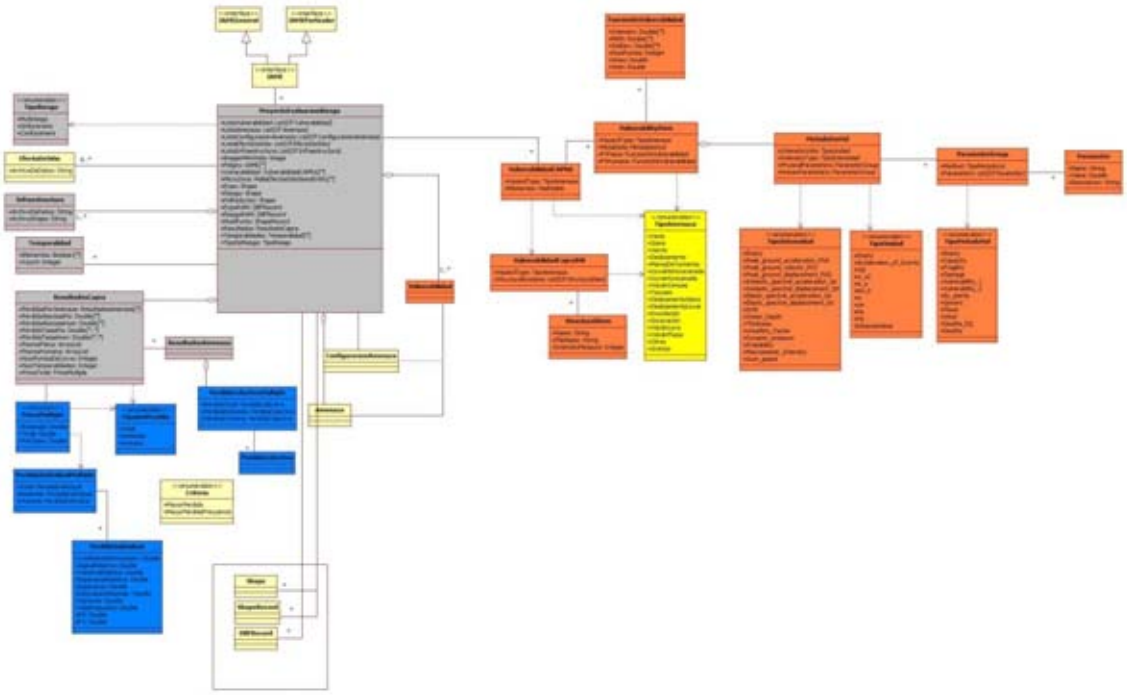


Figura 5-41 Diagrama de clases que muestra al modelo de dominio completo tras finalizar la reestructuración de código en capas y entidades de negocio (Elaboración propia, 2014). Esta figura se presenta a página completa en el anexo 2 de este trabajo para su mejor lectura.

VALIDACIÓN DE OBJETIVOS DE LA ARQUITECTURA

A continuación se presenta nuevamente la tabla 4-3 con lista de los objetivos planteados al arranque del diseño de la arquitectura propuesta para el sistema experto CAPRA. Se ha colocado a cada objetivo una breve indicación que valida la implementación y cumplimiento del mismo.

Clave	Objetivo	Validado
OB-01	Diseñar una arquitectura completa que sea implementada y verificada.	Sí. Se implementó y se realizó comprobación de funcionalidad.
OB-02	Soportar la misma funcionalidad de negocio de la versión 2.0 de CAPRA	Sí. Se comprobó resultados idénticos a la línea base funcional.
OB-03	Respetar los layouts de los archivos de amenaza, vulnerabilidad y elementos expuestos.	Sí. Los formatos de archivos de datos no sufrieron cambios.
OB-04	Reestructurar los componentes existentes en una arquitectura en capas y orientada al dominio.	Sí. Se implementó la arquitectura en capas y orientada al dominio.
OB-05	Reutilizar la interfaz gráfica de usuario de la versión 2.0	Sí. Se respetó la misma GUI sin cambios.
OB-06	Utilizar las tecnologías de Microsoft .Net y Visual Basic.Net como lenguaje de programación.	Sí. Se utilizó visual basic.net como lenguaje y Visual Studio 2010 como IDE de desarrollo.
OB-07	Aprovechar el poder de cómputo del equipo cliente.	Sí. Se implementó una arquitectura para aplicación de escritorio.
OB-08	Aprovechar que los usuarios cuentan con sistema operativo Windows	Sí. CAPRA 3.0 es ejecutable sobre Windows XP y 7
OB-09	Aprovechar que los usuarios cuentan con permisos para la instalación de CAPRA en sus equipos.	Sí. Se generó un paquete de instalación para aplicación de escritorio.

Tabla 5-4 Validación de objetivos de la arquitectura propuesta para CAPRA 3.0

En este último capítulo se centra en la evidencia y resultados de la implementación de la arquitectura diseñada en el capítulo 4. Donde dicha arquitectura se propuso a partir de los fundamentos teóricos de las disciplinas de sistemas expertos, arquitectura de software y evaluación de riesgos naturales presentadas en los primeros capítulos.

La implementación de la arquitectura se ejecutó de manera metódica mediante las etapas de preparación, implementación, comprobación de funcionalidad, actualización del diseño, generación del instalador y validación de los objetivos de la arquitectura.

Adicionalmente, para dar un mayor detalle de la implementación de la arquitectura se propuso un proceso para reestructuración de código en capas y entidades de negocio (figura 5-16). Tras la ejecución de este proceso se generó la nueva versión que cumple con los objetivos trazados en el diseño de la arquitectura del sistema experto CAPRA 3.0.

El código fuente resultado de la implementación se encuentra disponible en el servicio de alojamiento de archivos OneDrive mediante la siguiente url corta http://goo.gl/8wVe82

CAPÍTULO 6 CONCLUSIONES

En este trabajo de tesis se diseñó e implementó una arquitectura de software para el sistema experto de evaluación de riesgos naturales CAPRA versión 3.0. Las siguientes conclusiones derivan de ello:

- Se obtuvo una arquitectura de software sencilla y sólida con un diseño en capas y orientada al dominio tras la aplicación de un proceso de diseño basado en fundamentos teóricos de la arquitectura de software.
- La arquitectura implementada cumplió con los objetivos definidos en su fase de diseño.
- Se aporta al campo de migración de software un proceso para la reestructuración del código en capas y entidades de negocio (figura 5-16).
- La arquitectura implementada y la reestructuración realizada al código ofrecen mejoras a las características de modularidad, mantenibilidad, reutilización e incorporación de funcionalidad.
- Los lenguajes de programación orientados a objetos son una alternativa para el desarrollo de sistemas expertos.
- Con la versión 3.0 de CAPRA los usuarios no se ven afectados pues esta versión respeta cualidades de la versión anterior como layouts de los archivos de amenaza, vulnerabilidad y exposición, y mantiene la misma funcionalidad de negocio e interfaz gráfica.
- El código reestructurado permanece escrito en Visual Basic.Net; por tanto los programadores de CAPRA no se ven obligados a aprender un nuevo lenguaje de programación, ni otra IDE de desarrollo.
- La reestructuración del código fue parcial pero suficiente para demostrar que es posible separar en capas el código actual de CAPRA. Con este proceso realizado en la implementación de la arquitectura es posible reutilizar el 100% de su código, invirtiendo así un menor esfuerzo para migrarlo hacia la arquitectura de software diseñada.
- La versión 3.0 de CAPRA producto de la reestructuración y actualización de su arquitectura ofreció los mismos resultados al ejecutar los dos escenarios de cálculo de riesgo; sin embargo no debe considerarse como una versión totalmente productiva porque para ello se requiere un conjunto más diverso de datos y pruebas.
- Esta arquitectura de software se ofrece como un marco de desarrollo para que los expertos en evaluación de riesgos naturales continúen integrando el conocimiento generado en sus disciplinas.
- Esta arquitectura se ofrece como base para la construcción de futuras versiones de CAPRA orientadas a evolucionar hacia tecnologías como los dispositivos móviles y aplicaciones en la nube.

- La arquitectura de software diseñada es simple; por tanto puede ser utilizada como un punto de partida para el desarrollo de otros sistemas expertos con un mínimo de conocimiento en programación orientada a objetos.

Como conclusión general, se debe tener siempre presente que un evento natural puede producir pérdidas de vidas humanas y pérdidas económicas con efectos devastadores directos e indirectos sobre familias, ciudades o un país entero. Es por ello que todas las disciplinas de la ingeniería civil deben mantener su alto compromiso con la sociedad, tanto en el campo de la práctica como en la investigación.

Este trabajo pretende aportar una gota más en ese océano de conocimiento, al proporcionar las bases para la construcción de nuevas versiones del sistema experto CAPRA, ofreciendo con ello la continuidad del desarrollo, uso y evolución de esta herramienta de software con el objetivo de mitigar y prevenir desastres naturales.

Futuros desarrollos

La versión CAPRA 3.0 es producto de la reestructuración parcial del código de CAPRA 2.0 con base en la nueva arquitectura de software diseñada e implementada en este trabajo; por lo tanto, se propone continuar con la restructuración restante así como la implementación de esta misma arquitectura en los demás sistemas de la plataforma CAPRA: CRISIS2007, ERN-HURACAN, ERN-DESLIZAMIENTO, ERN-VULNERABILIDAD, ERN-VOLCÁN, ERN-LLUVIANH y ERN-INUNDACIÓN.

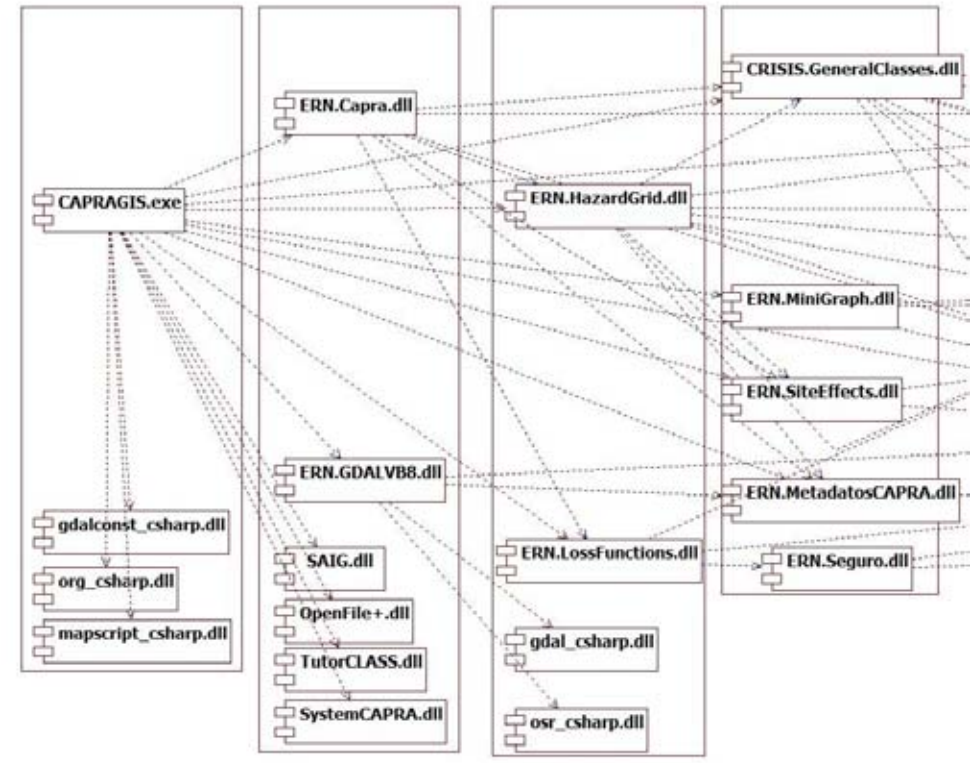
Otro desarrollo futuro, después de realizada la reestructuración completa, sería generar una versión de CAPRA para la tecnología de Cloud Computing (cómputo en la nube) en la modalidad de máquina virtual. Para lograrlo, bastaría con modificar el código de la capa de datos de modo tal que pueda acceder a los archivos de datos, ahora colocados en un Content Delivery Network (CDN) en lugar de la típica ruta C:\directorio\...\archivo.

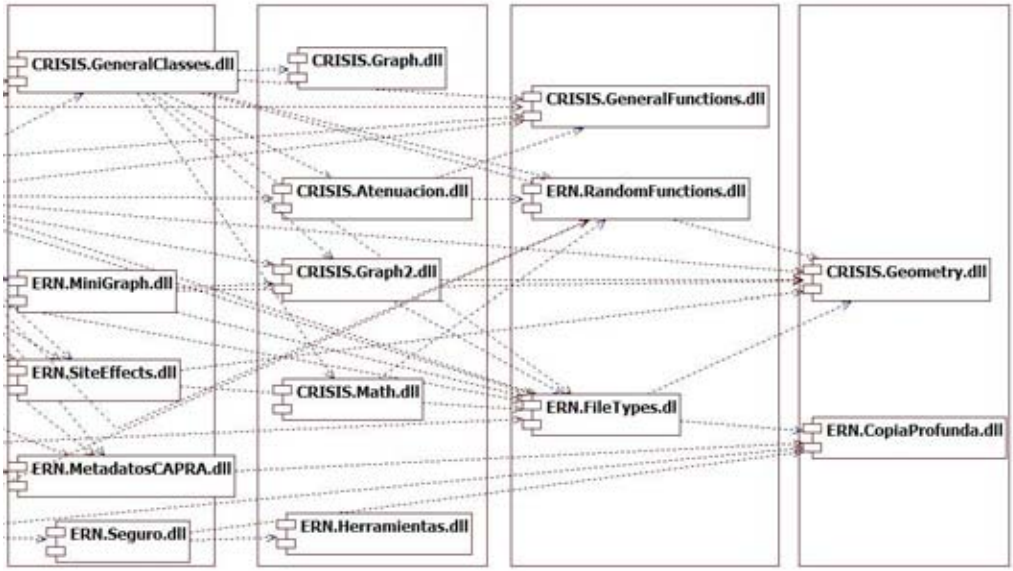
A partir de lo anterior, como meta siguiente se podría exponer la funcionalidad del cálculo del riesgo como un servicio en la nube (Cloud Service) al envolver la capa experta de CAPRA a través de componentes de tipo “Worker Role” en la plataforma para la nube de Microsoft Azure, donde dichos servicios sean consumidos por clientes móviles como tabletas o celulares inteligentes con interfaces gráficas simplificadas que permitan subir a la nube los archivos con datos para su cálculo desde cualquier sitio del planeta y/o visualizar resultados en cualquier otra ciudad del mundo en cuestión de segundos, con el apoyo de la enorme capacidad de cómputo y almacenamiento de la nube.

En ambos casos, se debe considerar que la implementación de la arquitectura y la restructuración del código de esos sistemas deben ser validadas por expertos en evaluación de riesgos naturales con el apoyo de un conjunto amplio de datos y casos de prueba diseñados a partir de las versiones actuales de estos sistemas.

ANEXO1

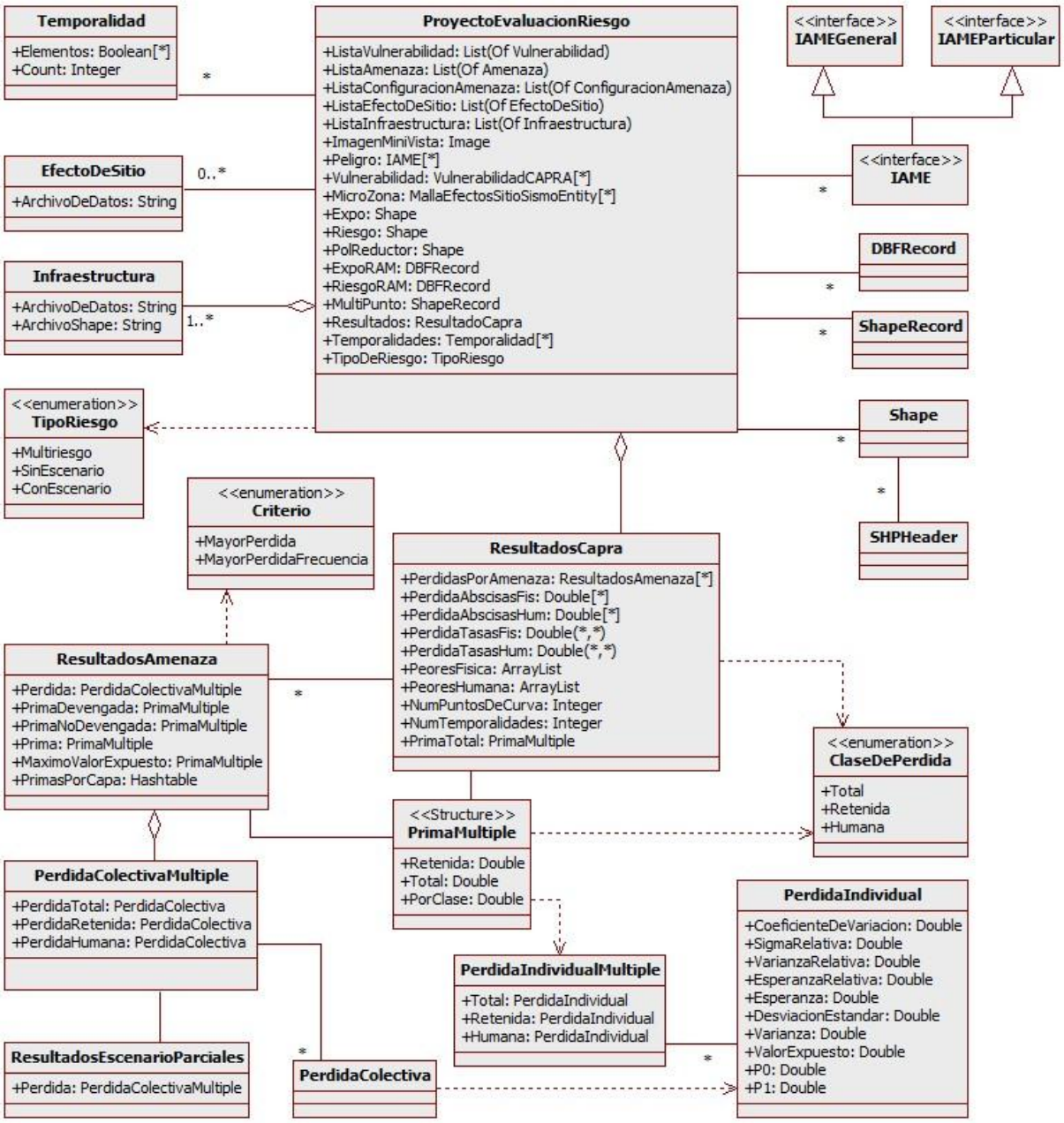
La siguiente figura es una réplica dividida de la figura 4-2 Diagrama de componentes del sistema CAPRA v2.0 (Elaboración propia, 2014).

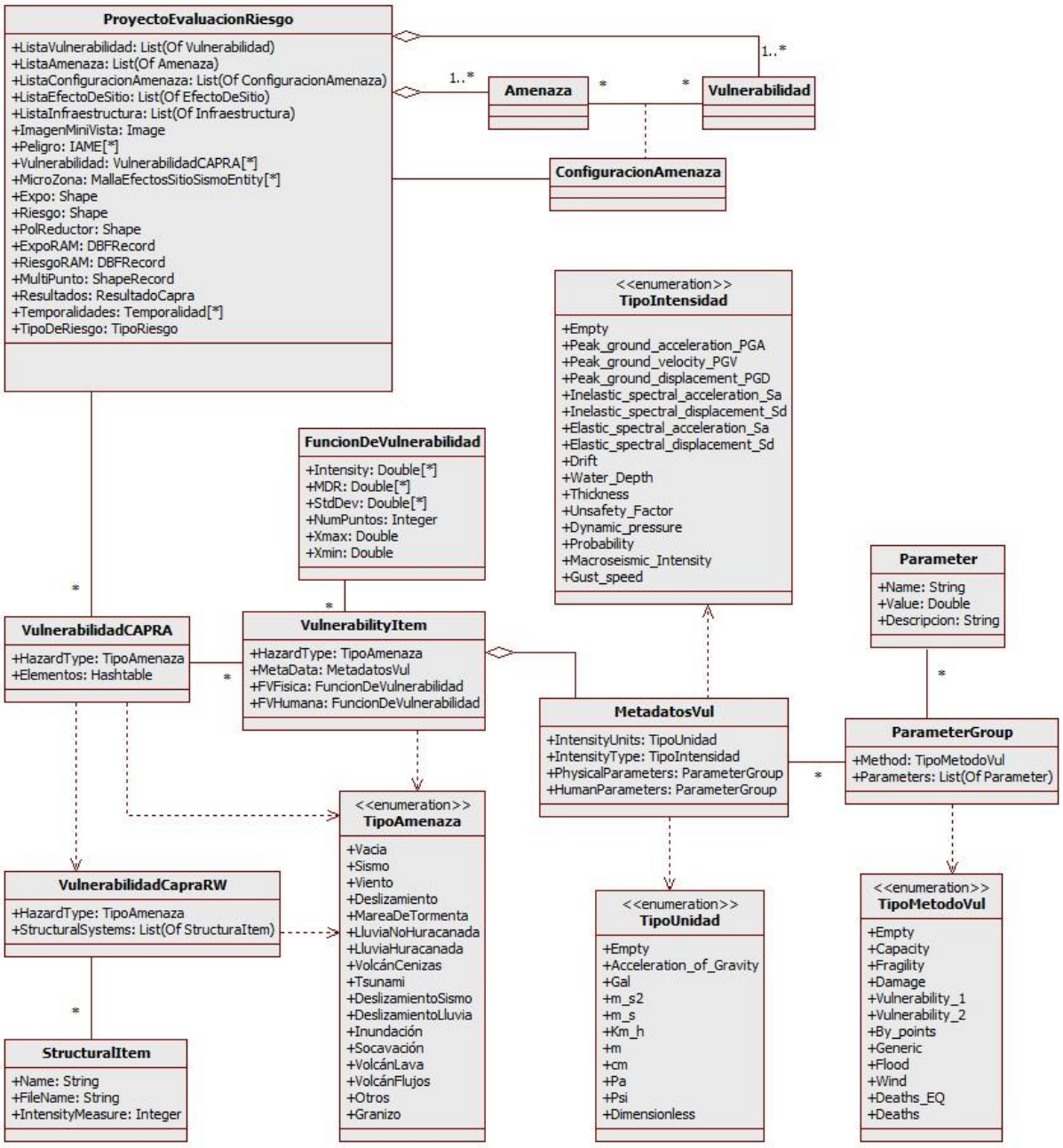




ANEXO 2

La siguiente figura es una réplica a dos páginas de la figura 5-41 Diagrama de clases que muestra al modelo de dominio completo tras finalizar la reestructuración de código en capas y entidades de negocio (Elaboración propia, 2014).




ProyectoEvaluacionRiesgo
+ListaVulnerabilidad: List(Of Vulnerabilidad)
+ListaAmenaza: List(Of Amenaza)
+ListaConfiguracionAmenaza: List(Of ConfiguracionAmenaza)
+ListaEfectoDeSitio: List(Of EfectoDeSitio)
+ListaInfraestructura: List(Of Infraestructura)
+ImagenMiniVista: Image
+Peligro: IAME[*]
+Vulnerabilidad: VulnerabilidadCAPRA[*]
+MicroZona: MallaEfectosSitioSismoEntity[*]
+Expo: Shape
+Riesgo: Shape
+PolReductor: Shape
+ExpoRAM: DBFRecord
+RiesgoRAM: DBFRecord
+MultiPunto: ShapeRecord
+Resultados: ResultadoCapra
+Temporalidades: Temporalidad[*]
+TipoDeRiesgo: TipoRiesgo
1..*
1..*
Amenaza
*
*
Vulnerabilidad
ConfiguracionAmenaza
<<enumeration>>
TipoIntensidad
+Empty
+Peak_ground_acceleration_PGA
+Peak_ground_velocity_PGV
+Peak_ground_displacement_PGD
+Inelastic_spectral_acceleration_Sa
+Inelastic_spectral_displacement_Sd
+Elastic_spectral_acceleration_Sa
+Elastic_spectral_displacement_Sd
+Drift
+Water_Depth
+Thickness
+Unsafety_Factor
+Dynamic_pressure
+Probability
+Macroseismic_Intensity
+Gust_speed
FuncionDeVulnerabilidad
+Intensity: Double[*]
+MDR: Double[*]
+StdDev: Double[*]
+NumPuntos: Integer
+Xmax: Double
+Xmin: Double
Parameter
+Name: String
+Value: Double
+Descripcion: String
VulnerabilidadCAPRA
+HazardType: TipoAmenaza
+Elementos: Hashtable
VulnerabilityItem
+HazardType: TipoAmenaza
+MetaData: MetadatosVul
+FVFisica: FuncionDeVulnerabilidad
+FVHumana: FuncionDeVulnerabilidad
MetadatosVul
+IntensityUnits: TipoUnidad
+IntensityType: TipoIntensidad
+PhysicalParameters: ParameterGroup
+HumanParameters: ParameterGroup
ParameterGroup
+Method: TipoMetodoVul
+Parameters: List(Of Parameter)
<<enumeration>>
TipoAmenaza
+Vacia
+Sismo
+Viento
+Deslizamiento
+MareaDeTormenta
+LluviaNoHuracanada
+LluviaHuracanada
+VolcánCenizas
+Tsunami
+DeslizamientoSismo
+DeslizamientoLluvia
+Inundación
+Socavación
+VolcánLava
+VolcánFlujos
+Otros
+Granizo
VulnerabilidadCapraRW
+HazardType: TipoAmenaza
+StructuralSystems: List(Of StructuralItem)
<<enumeration>>
TipoUnidad
+Empty
+Acceleration_of_Gravity
+Gal
+m_s2
+m_s
+Km_h
+m
+cm
+Pa
+Psi
+Dimensionless
<<enumeration>>
TipoMetodoVul
+Empty
+Capacity
+Fragility
+Damage
+Vulnerability_1
+Vulnerability_2
+By_points
+Generic
+Flood
+Wind
+Deaths_EQ
+Deaths
StructuralItem
+Name: String
+FileName: String
+IntensityMeasure: Integer

REFERENCIAS

Bass, L., & al, e. (2003). *Software Architecture in Practice.* Addison-Wesley.

Bauer, K., Büttel, I., Eberhard, L., Hälker, M., Lehner, H., Micholka, K., y otros. (1988). *Sistemas Expertos.* Barcelona: Dieter Nebendahl.

Benito, B., Cervera Bravo, J., Molina Palacios, S., Navarro Bernal, M., Doblas Lavigne, M. d., Martínez Díaz, J. J., y otros. (23 de noviembre de 2012). *Evaluación de la peligrosidad y el riesgo sísmico en Haití y aplicación al diseño sismorresistente.* Recuperado el 2013 de mayo de 16, de Archivo Digital Universidad Politécnica de Madrid: http://oa.upm.es/13999/1/Informe_SISMO-HAITI.pdf

CAPRA. (2012). *Tutoriales de CAPRA-GIS.* Recuperado el 4 de 5 de 2014, de CAPRA Probabilistic Risk Assessment Program: http://www.ecapra.org/es/capra-gis-0

CBO. (sep de 2005). *Macroeconomic and Budgetary Effects of Hurricanes Katrina y Rita.* Recuperado el 2013 de abril de 28, de Congressional Budget Office: http://www.cbo.gov/publication/17204

Chatain, J.-N., & Dussauchoy, A. (1988). *Sistemas Expertos Métodos y Herramientas.* Madrid: Parainfo.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., y otros. (2010). *Documenting Software Architectures: Views and Beyond(2nd Edition).* Westford: Addison-Wesley Professional.

de la Torre, C., Zorrilla, U., Calvarro, J., & Ramos, M. (2010). *Guía de Arquitectura de N-Capas orientada al Dominio con .Net 4.0.* España: Krasis PRESS.

DOF. (12 de may de 2012). *CIRCULAR Modificatoria 54/12 de la Unica de Seguros.* Recuperado el 9 de nov de 2013, de Diario Oficial de la Federación: http://dof.gob.mx/nota_detalle.php?codigo=5271415&fecha=05/10/2012

eCAPRA.org. (2012). *Software*. Recuperado el mayo de 2013, de Probabilistic Risk Assessment Program CAPRA: http://www.ecapra.org/es/software

Elaboración propia. (2014).

ERN. (2011). *INFORME TÉCNICO ERN-CAPRA-T1-1.* Recuperado el 18 de abril de 2013, de ecapra.org: http://www.ecapra.org/sites/default/files/documents/ERN-CAPRA-T1-1%20-%20Componentes%20Principales%20del%20Analisis%20de%20Riesgos.pdf

ERN Consorcio. (2011). *Metodología de Modelación Probabilista de Riesgos Naturales.* Recuperado el 18 de abril de 2013, de ecapra.org: http://www.ecapra.org/es/metodolog%C3%ADa-de-evaluaci%C3%B3n-probabilista-de-riesgos-naturales

FMI. (2013). *World Economic Outlook Database*. Recuperado el 25 de abril de 2013, de Fondo Monetario Internacional: http://www.imf.org/external/pubs/ft/weo/2012/02/weodata/weorept.aspx

Fontela, C. (2011). *UML: Modelado de software para profesionales.* Buenos Aires: Alfaomega grupo editor.

Fowler, M. (2002). *Patterns of Enterprise Application Architecture.* Indiana: Pearson Education.

Frenzel, L. E. (1989). *A fondo: sistemas expertos.* Madrid, España: ANAYA MULTIMEDIA.

Kruchten, P. (1995). Architectural Blueprints—The "4+1" View Model of Software Architecture. *IEEE Software 12*, 42-50.

Malveau, R., & Mowbray, T. J. (2004). *Software architect bootcamp.* Indianapolis: Pearson Education.

Martin, J., & Odell, J. (1997). *Métodos orientados a objetos: Conceptos fundamentales.* México: Prentice-Hall.

Microsoft patterns & practices. (2009). *Microsoft application architecture guide.* Microsoft.

Perez, J. (2008). *ULPGC.* Recuperado el 22 de junio de 2013, de http://www.iuma.ulpgc.es/~nunez/clases-micros-para-com/mpc0809-trabajos/mpc0809JavierPerezMatosupercomputadores_ppt.pdf

Reinoso, E., Jaimes, M. A., Ordaz, M., & Niño, M. (1 de enero de 2010). *Pérdidas en la infraestructura en México ante sismos y huracanes. Revista Digital Universitaria [en línea]. 1 de enero 2010, Vol. 11, No.1.* Recuperado el 4 de may0 de 2013, de http://www.revista.unam.mx/vol.11/num1/art05/int05/int05d.htm

Rittgen, P. (2007). *Enterprise modeling and computing with UML.* Idea Group Inc.

SEI. (18 de 3 de 2014). *Software Architecture*. Recuperado el 18 de 3 de 2014, de Software Engineering Institute: http://www.sei.cmu.edu/architecture/?location=secondary-nav&source=217167

Sommerville, I. (2004). *Software Engineering.* Pearson Addison-Wesley.

Taylor, R. N., Medvidovic, N., & Dashofy, E. M. (2010). *Software Architecture Foundations, Theory, and Practice.* USA: Wiley.

USGS. (2012). *Historic World Earthquakes*. Recuperado el 5 de mayo de 2013, de The United States Geological Survey: http://earthquake.usgs.gov/earthquakes/world/historical_mag.php